# Bin packing problems

## 23rd Belgian Mathematical Optimization Workshop

Silvano Martello

*DEI "Guglielmo Marconi", Alma Mater Studiorum Università di Bologna, Italy*

# The Bin Packing Problem

# The Bin Packing Problem

One of the most famous problems in combinatorial optimization.

Attacked with all main theoretical and practical tools.

# The Bin Packing Problem

One of the most famous problems in combinatorial optimization.

Attacked with all main theoretical and practical tools.

Packing problems have been studied since the Thirties (Kantorovich).

In 1961 Gilmore and Gomory introduced, for these problems, the concept of column generation.

# The Bin Packing Problem

One of the most famous problems in combinatorial optimization.

Attacked with all main theoretical and practical tools.

Packing problems have been studied since the Thirties (Kantorovich).

In 1961 Gilmore and Gomory introduced, for these problems, the concept of column generation.

The worst-case performance of approximation algorithms investigated since the early Seventies.

# The Bin Packing Problem

One of the most famous problems in combinatorial optimization.

Attacked with all main theoretical and practical tools.

Packing problems have been studied since the Thirties (Kantorovich).

In 1961 Gilmore and Gomory introduced, for these problems, the concept of column generation.

The worst-case performance of approximation algorithms investigated since the early Seventies.

Lower bounds and effective exact algorithms developed starting from the Eighties.

# The Bin Packing Problem

One of the most famous problems in combinatorial optimization.

Attacked with all main theoretical and practical tools.

Packing problems have been studied since the Thirties (Kantorovich).

In 1961 Gilmore and Gomory introduced, for these problems, the concept of column generation.

The worst-case performance of approximation algorithms investigated since the early Seventies.

Lower bounds and effective exact algorithms developed starting from the Eighties.

[Many heuristic and (starting from the Nineties) metaheuristic approaches.]

# The Bin Packing Problem

One of the most famous problems in combinatorial optimization.

Attacked with all main theoretical and practical tools.

Packing problems have been studied since the Thirties (Kantorovich).

In 1961 Gilmore and Gomory introduced, for these problems, the concept of column generation.

The worst-case performance of approximation algorithms investigated since the early Seventies.

Lower bounds and effective exact algorithms developed starting from the Eighties.

[Many heuristic and (starting from the Nineties) metaheuristic approaches.]

This talk will also introduce many basic general techniques for Combinatorial Optimization.

# The Bin Packing Problem

One of the most famous problems in combinatorial optimization.

Attacked with all main theoretical and practical tools.

Packing problems have been studied since the Thirties (Kantorovich).

In 1961 Gilmore and Gomory introduced, for these problems, the concept of column generation.

The worst-case performance of approximation algorithms investigated since the early Seventies.

Lower bounds and effective exact algorithms developed starting from the Eighties.

[Many heuristic and (starting from the Nineties) metaheuristic approaches.]

This talk will also introduce many basic general techniques for Combinatorial Optimization.

The field is still very active:

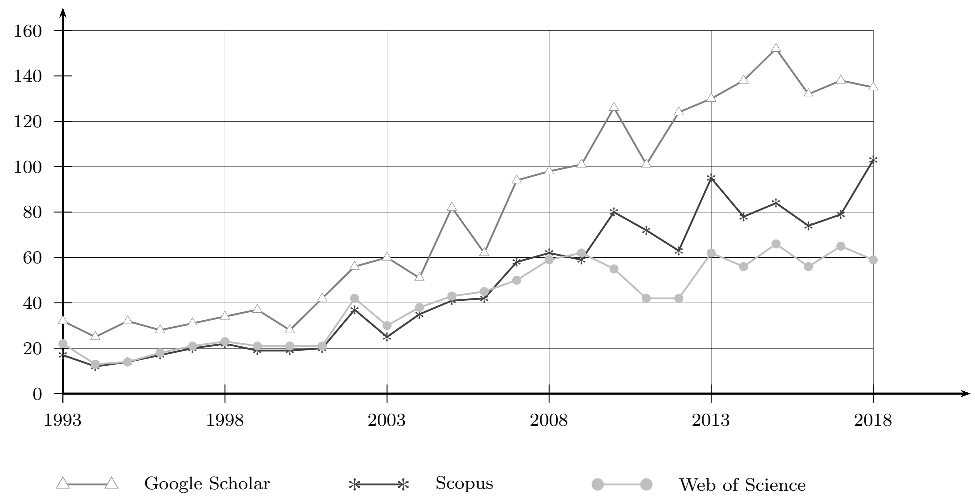



Figure 1: Number of papers dealing with bin packing and cutting stock problems, 1993-2018

# Contents

# Contents

# Definitions

# Definitions

**1)** Given $n$ *items*, each having an integer *weight* (or *size*) $w_j$ $(j = 1, \ldots, n)$, and an unlimited number of identical *bins* of integer *capacity* $c$,

## Definitions

**1)** Given $n$ *items*, each having an integer *weight* (or *size*) $w_j$ $(j = 1, \ldots, n)$, and an unlimited number of identical *bins* of integer *capacity* $c$,
**Bin Packing Problem (BPP)**: pack all the items into the minimum number of bins so that the total weight packed in any bin does not exceed the capacity.

# Definitions

**1)** Given $n$ *items*, each having an integer *weight* (or *size*) $w_j$ $(j = 1, \ldots, n)$, and an unlimited number of identical *bins* of integer *capacity* $c$,
**Bin Packing Problem (BPP)**: pack all the items into the minimum number of bins so that the total weight packed in any bin does not exceed the capacity.
We assume, with no loss of generality, that $0 < w_j < c$ for all $j$.

# Definitions

**1)** Given $n$ *items*, each having an integer *weight* (or *size*) $w_j$ $(j = 1, \ldots, n)$, and an unlimited number of identical *bins* of integer *capacity* $c$,
**Bin Packing Problem (BPP)**: pack all the items into the minimum number of bins so that the total weight packed in any bin does not exceed the capacity.
We assume, with no loss of generality, that $0 < w_j < c$ for all $j$.

---

Main application (**generalization**):

# Definitions

**1)** Given $n$ *items*, each having an integer *weight* (or *size*) $w_j$ $(j = 1, \ldots, n)$, and an unlimited number of identical *bins* of integer *capacity* $c$,
**Bin Packing Problem (BPP)**: pack all the items into the minimum number of bins so that the total weight packed in any bin does not exceed the capacity.
We assume, with no loss of generality, that $0 < w_j < c$ for all $j$.

---

Main application (**generalization**):

**2)** Given $m$ *item types*, each having an integer *weight* $w_j$ and an integer *demand* $d_j$ $(j = 1, \ldots, m)$, and
an unlimited number of identical *bins* of integer *capacity* $c$,

# Definitions

**1)** Given $n$ *items*, each having an integer *weight* (or *size*) $w_j$ $(j = 1, \ldots, n)$, and
an unlimited number of identical *bins* of integer *capacity* $c$,
**Bin Packing Problem (BPP)**: pack all the items into the minimum number of bins so that the total weight packed in any bin does not exceed the capacity.
We assume, with no loss of generality, that $0 < w_j < c$ for all $j$.

---

Main application (**generalization**):

**2)** Given $m$ *item types*, each having an integer *weight* $w_j$ and an integer *demand* $d_j$ $(j = 1, \ldots, m)$, and
an unlimited number of identical *bins* of integer *capacity* $c$,
**Cutting Stock Problem (CSP)**: produce (at least) $d_j$ copies of each item type $j$ using the minimum number of bins so that the total weight in any bin does not exceed the capacity.

# Definitions

**1)** Given $n$ *items*, each having an integer *weight* (or *size*) $w_j$ $(j = 1, \dots, n)$, and an unlimited number of identical *bins* of integer *capacity* $c$,
**Bin Packing Problem (BPP)**: pack all the items into the minimum number of bins so that the total weight packed in any bin does not exceed the capacity.
We assume, with no loss of generality, that $0 < w_j < c$ for all $j$.

---

Main application (**generalization**):

**2)** Given $m$ *item types*, each having an integer *weight* $w_j$ and an integer *demand* $d_j$ $(j = 1, \dots, m)$, and
an unlimited number of identical *bins* of integer *capacity* $c$,
**Cutting Stock Problem (CSP)**: produce (at least) $d_j$ copies of each item type $j$ using the minimum number of bins so that the total weight in any bin does not exceed the capacity.

Frequently interpreted as the process of *cutting pieces* (items) *from rolls of material* (bins).

# Definitions

**1)** Given $n$ *items*, each having an integer *weight* (or *size*) $w_j$ $(j = 1, \dots, n)$, and an unlimited number of identical *bins* of integer *capacity* $c$,
**Bin Packing Problem (BPP)**: pack all the items into the minimum number of bins so that the total weight packed in any bin does not exceed the capacity.
We assume, with no loss of generality, that $0 < w_j < c$ for all $j$.

---

Main application (**generalization**):

**2)** Given $m$ *item types*, each having an integer *weight* $w_j$ and an integer *demand* $d_j$ $(j = 1, \dots, m)$, and
an unlimited number of identical *bins* of integer *capacity* $c$,
**Cutting Stock Problem (CSP)**: produce (at least) $d_j$ copies of each item type $j$ using the minimum number of bins so that the total weight in any bin does not exceed the capacity.

Frequently interpreted as the process of *cutting pieces* (items) *from rolls of material* (bins).

---

**Real world** applications in
packing trucks with a given weight limit,

assigning commercials to station breaks

allocating memory in computers,

subproblems in more complex optimization problems ...

# Polynomial models (textbooks)

# Polynomial models (textbooks)

- Let $u$ be any upper bound on the minimum number of bins needed (e.g., approximate solution), assume that the potential bins are numbered as $1, \dots, u$.

## Polynomial models (textbooks)

- Let $u$ be any upper bound on the minimum number of bins needed (e.g., approximate solution), assume that the potential bins are numbered as $1, \ldots, u$.

$$y_i = \begin{cases} 1 & \text{if bin } i \text{ is used in the solution;} \\ 0 & \text{otherwise} \end{cases} \qquad (i = 1, \ldots, u),$$

$$x_{ij} = \begin{cases} 1 & \text{if item } j \text{ is packed into bin } i; \\ 0 & \text{otherwise} \end{cases} \qquad (i = 1, \ldots, u; j = 1, \ldots, n),$$

# Polynomial models (textbooks)

- Let $u$ be any upper bound on the minimum number of bins needed (e.g., approximate solution), assume that the potential bins are numbered as $1, \ldots, u$.

$$y_i = \begin{cases} 1 & \text{if bin } i \text{ is used in the solution;} \\ 0 & \text{otherwise} \end{cases} \qquad (i = 1, \ldots, u),$$

$$x_{ij} = \begin{cases} 1 & \text{if item } j \text{ is packed into bin } i; \\ 0 & \text{otherwise} \end{cases} \qquad (i = 1, \ldots, u; j = 1, \ldots, n),$$

- *Integer Linear Program* (ILP) for the **BPP** (Martello and Toth, 1990)

$$\min \quad \sum_{i=1}^{u} y_i \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_{ij} \le c y_i \quad (i = 1, \ldots, u), \tag{2}$$

$$\sum_{i=1}^{u} x_{ij} = 1 \qquad (j = 1, \ldots, n), \tag{3}$$

$$y_i \in \{0, 1\} \qquad (i = 1, \ldots, u), \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad (i = 1, \ldots, u; j = 1, \ldots, n). \tag{5}$$

# Polynomial models (textbooks)

- Let $u$ be any upper bound on the minimum number of bins needed (e.g., approximate solution), assume that the potential bins are numbered as $1, \ldots, u$.

$$y_i = \begin{cases} 1 & \text{if bin } i \text{ is used in the solution;} \\ 0 & \text{otherwise} \end{cases} \qquad (i = 1, \ldots, u),$$

$$x_{ij} = \begin{cases} 1 & \text{if item } j \text{ is packed into bin } i; \\ 0 & \text{otherwise} \end{cases} \qquad (i = 1, \ldots, u; j = 1, \ldots, n),$$

- *Integer Linear Program* (ILP) for the **BPP** (Martello and Toth, 1990)

$$\min \quad \sum_{i=1}^{u} y_i \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_{ij} \leq c y_i \quad (i = 1, \ldots, u), \tag{2}$$

$$\sum_{i=1}^{u} x_{ij} = 1 \quad (j = 1, \ldots, n), \tag{3}$$

$$y_i \in \{0, 1\} \quad (i = 1, \ldots, u), \tag{4}$$

$$x_{ij} \in \{0, 1\} \quad (i = 1, \ldots, u; j = 1, \ldots, n). \tag{5}$$

- **Polynomial number of variables and constraints**

# Polynomial models (cont'd)

## Polynomial models (cont'd)

- $u$ and $y_i$ as before;

## Polynomial models (cont'd)

- $u$ and $y_i$ as before;

  $\xi_{ij}$ = number of items of type $j$ packed into bin $i$ $(i = 1, \ldots, u; j = 1, \ldots, m)$.

## Polynomial models (cont'd)

- $u$ and $y_i$ as before;

  $\xi_{ij}$ = number of items of type $j$ packed into bin $i$ $(i = 1, \ldots, u; j = 1, \ldots, m)$.

- *Integer Linear Program* (ILP) for the **CSP**

$$\min \quad \sum_{i=1}^{u} y_i \tag{6}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} w_j \xi_{ij} \le c y_i \quad (i = 1, \ldots, u), \tag{7}$$

$$\sum_{i=1}^{u} \xi_{ij} = d_j \qquad (j = 1, \ldots, m), \text{ [or } \ge d_j \text{(equivalent)]} \tag{8}$$

$$y_i \in \{0, 1\} \qquad (i = 1, \ldots, u), \tag{9}$$

$$\xi_{ij} \ge 0, \text{ integer} \qquad (i = 1, \ldots, u; j = 1, \ldots, m). \tag{10}$$

# Polynomial models (cont'd)

- $u$ and $y_i$ as before;

  $\xi_{ij}$ = number of items of type $j$ packed into bin $i$ $(i = 1, \ldots, u; j = 1, \ldots, m)$.

- *Integer Linear Program* (ILP) for the **CSP**

$$\min \sum_{i=1}^{u} y_i \tag{6}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} w_j \xi_{ij} \le c y_i \quad (i = 1, \ldots, u), \tag{7}$$

$$\sum_{i=1}^{u} \xi_{ij} = d_j \qquad (j = 1, \ldots, m), \text{ [or } \ge d_j \text{(equivalent)]} \tag{8}$$

$$y_i \in \{0, 1\} \qquad (i = 1, \ldots, u), \tag{9}$$

$$\xi_{ij} \ge 0, \text{integer} \qquad (i = 1, \ldots, u; j = 1, \ldots, m). \tag{10}$$

BPP = special case of the CSP in which $d_j = 1$ for all $j$;

## Polynomial models (cont'd)

- $u$ and $y_i$ as before;

  $\xi_{ij}$ = number of items of type $j$ packed into bin $i$ $(i = 1, \dots, u; j = 1, \dots, m)$.

- *Integer Linear Program* (ILP) for the **CSP**

$$\min \sum_{i=1}^{u} y_i \tag{6}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} w_j \xi_{ij} \le c y_i \quad (i = 1, \dots, u), \tag{7}$$

$$\sum_{i=1}^{u} \xi_{ij} = d_j \quad (j = 1, \dots, m), \text{ [or } \ge d_j \text{(equivalent)]} \tag{8}$$

$$y_i \in \{0, 1\} \quad (i = 1, \dots, u), \tag{9}$$

$$\xi_{ij} \ge 0, \text{ integer} \quad (i = 1, \dots, u; j = 1, \dots, m). \tag{10}$$

BPP = special case of the CSP in which $d_j = 1$ for all $j$;

CSP = a BPP in which the item set includes $d_j$ copies of each item type $j$.

# Polynomial models (cont'd)

- $u$ and $y_i$ as before;

  $\xi_{ij}$ = number of items of type $j$ packed into bin $i$ $(i = 1, \dots, u; j = 1, \dots, m)$.

- *Integer Linear Program* (ILP) for the **CSP**

$$\min \quad \sum_{i=1}^{u} y_i \tag{6}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} w_j \xi_{ij} \le c y_i \quad (i = 1, \dots, u), \tag{7}$$

$$\sum_{i=1}^{u} \xi_{ij} = d_j \quad (j = 1, \dots, m), \text{ [or } \ge d_j \text{(equivalent)]} \tag{8}$$

$$y_i \in \{0, 1\} \quad (i = 1, \dots, u), \tag{9}$$

$$\xi_{ij} \ge 0, \text{integer} \quad (i = 1, \dots, u; j = 1, \dots, m). \tag{10}$$

BPP = special case of the CSP in which $d_j = 1$ for all $j$;

CSP = a BPP in which the item set includes $d_j$ copies of each item type $j$.

The BPP (and hence the CSP) has been proved to be $\mathcal{NP}$-hard in the strong sense (Garey and Johnson, 1979: transformation from 3-Partition).

# Upper and lower bounds

## Upper and lower bounds

- We will normally refer to the **BPP** (unless otherwise specified).
- **Worst-case performance**
- Given a minimization problem and an approximation algorithm $A$, let
  - $A(I)$ = solution value provided by $A$ for an instance $I$;

# Upper and lower bounds

- We will normally refer to the **BPP** (unless otherwise specified).
- **Worst-case performance**
- Given a minimization problem and an approximation algorithm $A$, let
  - $A(I)$ = solution value provided by $A$ for an instance $I$;
  - $OPT(I)$ = optimal solution value for an instance $I$.

## Upper and lower bounds

- We will normally refer to the **BPP** (unless otherwise specified).
- **Worst-case performance**
- Given a minimization problem and an approximation algorithm $A$, let
  - $A(I)$ = solution value provided by $A$ for an instance $I$;
  - $OPT(I)$ = optimal solution value for an instance $I$.

  **Then**

  **Worst-case performance ratio (WCPR)** of $A =$

  smallest real number $\overline{r}(A) > 1$ such that $A(I)/OPT(I) \leq \overline{r}(A)$ for all instances $I$, i.e.,

$$\overline{r}(A) = \sup_I \{A(I)/OPT(I)\}.$$

# Upper and lower bounds

- We will normally refer to the **BPP** (unless otherwise specified).
- **Worst-case performance**
- Given a minimization problem and an approximation algorithm $A$, let
  - $A(I)$ = solution value provided by $A$ for an instance $I$;
  - *OPT*$(I)$ = optimal solution value for an instance $I$.

  **Then**

  **Worst-case performance ratio (WCPR)** of $A$ =

  smallest real number $\overline{r}(A) > 1$ such that $A(I)/OPT(I) \leq \overline{r}(A)$ for all instances $I$, i.e.,

$$\overline{r}(A) = \sup_{I}\{A(I)/OPT(I)\}.$$

- Given a minimization problem and a lower bounding procedure $L$, let
  - $L(I)$ = lower bound provided by $L$ for an instance $I$.

## Upper and lower bounds

- We will normally refer to the **BPP** (unless otherwise specified).
- **Worst-case performance**
- Given a minimization problem and an approximation algorithm $A$, let
  - $A(I)$ = solution value provided by $A$ for an instance $I$;
  - $OPT(I)$ = optimal solution value for an instance $I$.

  **Then**

  **Worst-case performance ratio (WCPR)** of $A$ =

  smallest real number $\overline{r}(A) > 1$ such that $A(I)/OPT(I) \leq \overline{r}(A)$ for all instances $I$, i.e.,

$$\overline{r}(A) = \sup_I \{A(I)/OPT(I)\}.$$

- Given a minimization problem and a lower bounding procedure $L$, let
  - $L(I)$ = lower bound provided by $L$ for an instance $I$.

  **Then**

  **Worst-case performance ratio (WCPR)** of $L$ =

  largest real number $\underline{r}(L) < 1$ such that $L(I)/OPT(I) \geq \underline{r}(L)$ for all instances $I$, i.e.,

$$\underline{r}(L) = \inf_I \{L(I)/OPT(I)\}.$$

# Approximation algorithms

# Approximation algorithms

- Seminal results: David Johnson's PhD thesis, 1973.
- Huge literature (specific surveys, $\sim$ 200 references).

## Approximation algorithms

- Seminal results: David Johnson's PhD thesis, 1973.
- Huge literature (specific surveys, $\sim$ 200 references).
- Two main families:
  - **On-line algorithms:** sequentially assign items to bins, in the order encountered in input, without knowledge of items not yet packed.

# Approximation algorithms

- Seminal results: David Johnson's PhD thesis, 1973.
- Huge literature (specific surveys, $\sim$ 200 references).
- Two main families:
  - **On-line algorithms:** sequentially assign items to bins, in the order encountered in input, without knowledge of items not yet packed.
  - **Off-line algorithms:** all items are known in advance, and are available for sorting, preprocessing, grouping, etc.

# Approximation algorithms

- Seminal results: David Johnson's PhD thesis, 1973.
- Huge literature (specific surveys, $\sim$ 200 references).
- Two main families:
  - **On-line algorithms:** sequentially assign items to bins, in the order encountered in input, without knowledge of items not yet packed.
  - **Off-line algorithms:** all items are known in advance, and are available for sorting, preprocessing, grouping, etc.
- Many other (less relevant) families:
  - semi on-line,
  - bounded space,
  - open-end,
  - conservative,
  - re-pack,
  - dynamic,
  - ...

# On-line algorithms

# On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;

## On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$

# On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).

## On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).
- **First-Fit (FF)**: pack the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin.

# On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).
- **First-Fit (FF)**: pack the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin.
  time complexity: trivial implementation: $O(n^2)$.

# On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).
- **First-Fit (FF)**: pack the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin.
  time complexity: trivial implementation: $O(n^2)$. With special data structures: $O(n \log n)$.

# On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).
- **First-Fit (FF)**: pack the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin.
  time complexity: trivial implementation: $O(n^2)$. With special data structures: $O(n \log n)$.
- **Best-Fit (BF)**: pack the next item into the feasible bin (if any) where it fits by leaving the smallest residual space, or into a new one if no open bin can accommodate it;

# On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).
- **First-Fit (FF)**: pack the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin.
  time complexity: trivial implementation: $O(n^2)$. With special data structures: $O(n \log n)$.
- **Best-Fit (BF)**: pack the next item into the feasible bin (if any) where it fits by leaving the smallest residual space, or into a new one if no open bin can accommodate it;
  time complexity: same as FF.

# On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).
- **First-Fit (FF)**: pack the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin.
  time complexity: trivial implementation: $O(n^2)$. With special data structures: $O(n \log n)$.
- **Best-Fit (BF)**: pack the next item into the feasible bin (if any) where it fits by leaving the smallest residual space, or into a new one if no open bin can accommodate it;
  time complexity: same as FF.

- **Numerical example:**

  $n = 12$, $c = 100$, $(w_j) =$ ( 50 3 48 53 53 4 3 41 23 20 52 49 )

  **NF:** $\{50 \quad 3\}$, $\{48\}$, $\{53\}$, $\{53 \quad 4 \quad 3\}$, $\{41 \quad 23 \quad 20\}$, $\{52\}$, $\{49\}$ **7 bins**

## On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).
- **First-Fit (FF)**: pack the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin.
  time complexity: trivial implementation: $O(n^2)$. With special data structures: $O(n \log n)$.
- **Best-Fit (BF)**: pack the next item into the feasible bin (if any) where it fits by leaving the smallest residual space, or into a new one if no open bin can accommodate it;
  time complexity: same as FF.
- **Numerical example:**

  $n = 12$, $c = 100$, $(w_j) =$ ( 50 3 48 53 53 4 3 41 23 20 52 49 )

  **NF:** $\{50 \quad 3\}$, $\{48\}$, $\{53\}$, $\{53 \quad 4 \quad 3\}$, $\{41 \quad 23 \quad 20\}$, $\{52\}$, $\{49\}$ **7 bins**

  **FF:** $\{50 \quad 3 \quad 4 \quad 3 \quad 23\}$, $\{48 \quad 41\}$, $\{53 \quad 20\}$, $\{53\}$, $\{52\}$, $\{49\}$ **6 bins**

# On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).
- **First-Fit (FF)**: pack the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin.
  time complexity: trivial implementation: $O(n^2)$. With special data structures: $O(n \log n)$.
- **Best-Fit (BF)**: pack the next item into the feasible bin (if any) where it fits by leaving the smallest residual space, or into a new one if no open bin can accommodate it;
  time complexity: same as FF.

- **Numerical example:**

  $n = 12$, $c = 100$, $(w_j) =$ ( 50 3 48 53 53 4 3 41 23 20 52 49 )

  **NF:** $\{50 \quad 3\}$, $\{48\}$, $\{53\}$, $\{53 \quad 4 \quad 3\}$, $\{41 \quad 23 \quad 20\}$, $\{52\}$, $\{49\}$ **7 bins**

  **FF:** $\{50 \quad 3 \quad 4 \quad 3 \quad 23\}$, $\{48 \quad 41\}$, $\{53 \quad 20\}$, $\{53\}$, $\{52\}$, $\{49\}$ **6 bins**

  **BF:** $\{50 \quad 3 \quad 4 \quad 3 \quad 23\}$, $\{48 \quad 52\}$, $\{53 \quad 41\}$, $\{53 \quad 20\}$, $\{49\}$ **5 bins**

# On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).
- **First-Fit (FF)**: pack the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin.
  time complexity: trivial implementation: $O(n^2)$. With special data structures: $O(n \log n)$.
- **Best-Fit (BF)**: pack the next item into the feasible bin (if any) where it fits by leaving the smallest residual space, or into a new one if no open bin can accommodate it;
  time complexity: same as FF.
- **Numerical example:**

  $n = 12$, $c = 100$, $(w_j) =$ ( 50 3 48 53 53 4 3 41 23 20 52 49 )

  **NF:** $\{50 \quad 3\}$, $\{48\}$, $\{53\}$, $\{53 \quad 4 \quad 3\}$, $\{41 \quad 23 \quad 20\}$, $\{52\}$, $\{49\}$ **7 bins**

  **FF:** $\{50 \quad 3 \quad 4 \quad 3 \quad 23\}$, $\{48 \quad 41\}$, $\{53 \quad 20\}$, $\{53\}$, $\{52\}$, $\{49\}$ **6 bins**

  **BF:** $\{50 \quad 3 \quad 4 \quad 3 \quad 23\}$, $\{48 \quad 52\}$, $\{53 \quad 41\}$, $\{53 \quad 20\}$, $\{49\}$ **5 bins**
- The **exact WCPR of FF and BF** has been an open problem for forty years,

# On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).
- **First-Fit (FF)**: pack the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin.
  time complexity: trivial implementation: $O(n^2)$. With special data structures: $O(n \log n)$.
- **Best-Fit (BF)**: pack the next item into the feasible bin (if any) where it fits by leaving the smallest residual space, or into a new one if no open bin can accommodate it;
  time complexity: same as FF.

- **Numerical example:**

  $n = 12$, $c = 100$, $(w_j) =$ ( 50 3 48 53 53 4 3 41 23 20 52 49 )

  **NF:** $\{50 \quad 3\}$, $\{48\}$, $\{53\}$, $\{53 \quad 4 \quad 3\}$, $\{41 \quad 23 \quad 20\}$, $\{52\}$, $\{49\}$ **7 bins**

  **FF:** $\{50 \quad 3 \quad 4 \quad 3 \quad 23\}$, $\{48 \quad 41\}$, $\{53 \quad 20\}$, $\{53\}$, $\{52\}$, $\{49\}$ **6 bins**

  **BF:** $\{50 \quad 3 \quad 4 \quad 3 \quad 23\}$, $\{48 \quad 52\}$, $\{53 \quad 41\}$, $\{53 \quad 20\}$, $\{49\}$ **5 bins**

- The **exact WCPR of FF and BF** has been an open problem for forty years, until recently (2014) Dósa and Sgall proved that $\overline{r}(FF) = \overline{r}(BF) = \frac{17}{10}$.

## On-line algorithms

- **Next-Fit (NF)**: pack the next item into the current bin if it fits, or into a new bin (which becomes the current one) if it doesn't;
  time complexity: $O(n)$;
  worst-case: $\overline{r}(NF) = 2$ (Hint: the contents of two consecutive bins is $> c$).
- **First-Fit (FF)**: pack the next item into the lowest indexed bin where it fits, or into a new bin if it does not fit in any open bin.
  time complexity: trivial implementation: $O(n^2)$. With special data structures: $O(n \log n)$.
- **Best-Fit (BF)**: pack the next item into the feasible bin (if any) where it fits by leaving the smallest residual space, or into a new one if no open bin can accommodate it;
  time complexity: same as FF.
- **Numerical example:**

  $n = 12$, $c = 100$, $(w_j) = ($ 50 3 48 53 53 4 3 41 23 20 52 49 $)$

  **NF:** $\{50 \quad 3\}, \{48\}, \{53\}, \{53 \quad 4 \quad 3\}, \{41 \quad 23 \quad 20\}, \{52\}, \{49\}$ **7 bins**

  **FF:** $\{50 \quad 3 \quad 4 \quad 3 \quad 23\}, \{48 \quad 41\}, \{53 \quad 20\}, \{53\}, \{52\}, \{49\}$ **6 bins**

  **BF:** $\{50 \quad 3 \quad 4 \quad 3 \quad 23\}, \{48 \quad 52\}, \{53 \quad 41\}, \{53 \quad 20\}, \{49\}$ **5 bins**
- The **exact WCPR of FF and BF** has been an open problem for forty years, until recently (2014) Dósa and Sgall proved that $\overline{r}(FF) = \overline{r}(BF) = \frac{17}{10}$.
- Other algorithms: **Worse-Fit** (**WF**, leave the largest residual space), Any-Fit, Almost Any-Fit, Bounded space, Next-$k$-Fit, Harmonic-Fit, Refined First-Fit, Modified Harmonic-Fit, ...

# Off-line algorithms

# Off-line algorithms

- Most of the classical on-line algorithms achieve their worst-case performance when the items are packed in increasing order of size or if small and large items are merged, and hence

# Off-line algorithms

- Most of the classical on-line algorithms achieve their worst-case performance when the items are packed in increasing order of size or if small and large items are merged, and hence
- main off-line category: sort the items in **decreasing order** of size (time $O(n \log n)$).
- **Next-Fit Decreasing**
  time complexity: $O(n \log n)$;

# Off-line algorithms

- Most of the classical on-line algorithms achieve their worst-case performance when the items are packed in increasing order of size or if small and large items are merged, and hence
- main off-line category: sort the items in **decreasing order** of size (time $O(n \log n)$).
- **Next-Fit Decreasing**
  time complexity: $O(n \log n)$;
  Exact worst-case unknown. It has been proved that it is not more than $\frac{7}{4}$.

## Off-line algorithms

- Most of the classical on-line algorithms achieve their worst-case performance when the items are packed in increasing order of size or if small and large items are merged, and hence
- main off-line category: sort the items in **decreasing order** of size (time $O(n \log n)$).
- **Next-Fit Decreasing**
  time complexity: $O(n \log n)$;
  Exact worst-case unknown. It has been proved that it is not more than $\frac{7}{4}$.
- **First-Fit Decreasing**; time complexity: $O(n \log n)$; worst-case: $\overline{r}(FFD) = \frac{3}{2}$.

## Off-line algorithms

- Most of the classical on-line algorithms achieve their worst-case performance when the items are packed in increasing order of size or if small and large items are merged, and hence
- main off-line category: sort the items in **decreasing order** of size (time $O(n \log n)$).
- **Next-Fit Decreasing**
  time complexity: $O(n \log n)$;
  Exact worst-case unknown. It has been proved that it is not more than $\frac{7}{4}$.
- **First-Fit Decreasing**; time complexity: $O(n \log n)$; worst-case: $\overline{r}(FFD) = \frac{3}{2}$.
- **Best-Fit Decreasing (BFD)**; time complexity: $O(n \log n)$; worst-case: $\overline{r}(BFD) = \frac{3}{2}$.

# Off-line algorithms

- Most of the classical on-line algorithms achieve their worst-case performance when the items are packed in increasing order of size or if small and large items are merged, and hence
- main off-line category: sort the items in **decreasing order** of size (time $O(n \log n)$).
- **Next-Fit Decreasing**
  time complexity: $O(n \log n)$;
  Exact worst-case unknown. It has been proved that it is not more than $\frac{7}{4}$.
- **First-Fit Decreasing**; time complexity: $O(n \log n)$; worst-case: $\overline{r}(FFD) = \frac{3}{2}$.
- **Best-Fit Decreasing (BFD)**; time complexity: $O(n \log n)$; worst-case: $\overline{r}(BFD) = \frac{3}{2}$.
- **Numerical example** (resumed):

  $n = 12$, $c = 100$, $(w_j) =$ ( 50 3 48 53 53 4 3 41 23 20 52 49 );
  Sorted items: $(w_j) =$ ( 53 53 52 50 49 48 41 23 20 4 3 3 );

  **NFD:** $\{53\}$, $\{53\}$, $\{52\}$, $\{50 \quad 49\}$, $\{48 \quad 41\}$, $\{23 \quad 20 \quad 4 \quad 3 \quad 3\}$ **6 bins**

# Off-line algorithms

- Most of the classical on-line algorithms achieve their worst-case performance when the items are packed in increasing order of size or if small and large items are merged, and hence
- main off-line category: sort the items in **decreasing order** of size (time $O(n \log n)$).
- **Next-Fit Decreasing**
  time complexity: $O(n \log n)$;
  Exact worst-case unknown. It has been proved that it is not more than $\frac{7}{4}$.
- **First-Fit Decreasing**; time complexity: $O(n \log n)$; worst-case: $\overline{r}(FFD) = \frac{3}{2}$.
- **Best-Fit Decreasing (BFD)**; time complexity: $O(n \log n)$; worst-case: $\overline{r}(BFD) = \frac{3}{2}$.
- **Numerical example** (resumed):

  $n = 12$, $c = 100$, $(w_j) =$ ( 50 3 48 53 53 4 3 41 23 20 52 49 );
  Sorted items: $(w_j) =$ ( 53 53 52 50 49 48 41 23 20 4 3 3 );

  **NFD:** $\{53\}$, $\{53\}$, $\{52\}$, $\{50 \quad 49\}$, $\{48 \quad 41\}$, $\{23 \quad 20 \quad 4 \quad 3 \quad 3\}$ **6 bins**

  **FFD:** $\{53 \quad 41 \quad 4\}$, $\{53 \quad 23 \quad 20 \quad 3\}$, $\{52 \quad 48\}$, $\{50 \quad 49\}$, $\{ 3\}$ **5 bins**

# Off-line algorithms

- Most of the classical on-line algorithms achieve their worst-case performance when the items are packed in increasing order of size or if small and large items are merged, and hence
- main off-line category: sort the items in **decreasing order** of size (time $O(n \log n)$).
- **Next-Fit Decreasing**
  time complexity: $O(n \log n)$;
  Exact worst-case unknown. It has been proved that it is not more than $\frac{7}{4}$.
- **First-Fit Decreasing**; time complexity: $O(n \log n)$; worst-case: $\overline{r}(FFD) = \frac{3}{2}$.
- **Best-Fit Decreasing (BFD)**; time complexity: $O(n \log n)$; worst-case: $\overline{r}(BFD) = \frac{3}{2}$.
- **Numerical example** (resumed):

  $n = 12$, $c = 100$, $(w_j) =$ ( 50 3 48 53 53 4 3 41 23 20 52 49 );
  Sorted items: $(w_j) =$ ( 53 53 52 50 49 48 41 23 20 4 3 3 );

  **NFD:** $\{53\}$, $\{53\}$, $\{52\}$, $\{50 \quad 49\}$, $\{48 \quad 41\}$, $\{23 \quad 20 \quad 4 \quad 3 \quad 3\}$ **6 bins**

  **FFD:** $\{53 \quad 41 \quad 4\}$, $\{53 \quad 23 \quad 20 \quad 3\}$, $\{52 \quad 48\}$, $\{50 \quad 49\}$, $\{\ 3\}$ **5 bins**

  **BFD:** $\{53 \quad 41 \quad 3 \quad 3\}$, $\{53 \quad 23 \quad 20 \quad 4\}$, $\{52 \quad 48\}$, $\{50 \quad 49\}$, **4 bins, optimum**

# Best polynomially achievable worst-case performance

## Best polynomially achievable worst-case performance

- Worst-case of FFD and BFD: $\overline{r}(FFD) = \overline{r}(BFD) = \frac{3}{2}$.
- Can we find a better algorithm?

## Best polynomially achievable worst-case performance

- Worst-case of FFD and BFD: $\overline{r}(FFD) = \overline{r}(BFD) = \frac{3}{2}$.
- Can we find a better algorithm? **Bad news:**

# Best polynomially achievable worst-case performance

- Worst-case of FFD and BFD: $\overline{r}(FFD) = \overline{r}(BFD) = \frac{3}{2}$.
- Can we find a better algorithm? **Bad news:**
- **No polynomial-time approximation algorithm for the BPP can have a WCPR smaller than $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$.**
- PARTITION problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$?

## Best polynomially achievable worst-case performance

- Worst-case of FFD and BFD: $\overline{r}(FFD) = \overline{r}(BFD) = \frac{3}{2}$.
- Can we find a better algorithm? **Bad news:**
- **No polynomial-time approximation algorithm for the BPP can have a WCPR smaller than $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$.**
- PARTITION problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$?

  PARTITION is $\mathcal{NP}$-complete.

## Best polynomially achievable worst-case performance

- Worst-case of FFD and BFD: $\overline{r}(FFD) = \overline{r}(BFD) = \frac{3}{2}$.
- Can we find a better algorithm? **Bad news:**
- **No polynomial-time approximation algorithm for the BPP can have a WCPR smaller than $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$.**
- PARTITION problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$?

  PARTITION is $\mathcal{NP}$-complete.

  Assume a polynomial-time approximation algorithm $A$ for the BPP exists such that $OPT(I) > \frac{2}{3} A(I)$ for all instances $I$.

## Best polynomially achievable worst-case performance

- Worst-case of FFD and BFD: $\overline{r}(FFD) = \overline{r}(BFD) = \frac{3}{2}$.
- Can we find a better algorithm? **Bad news:**
- **No polynomial-time approximation algorithm for the BPP can have a WCPR smaller than $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$.**
- PARTITION problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$?

  PARTITION is $\mathcal{NP}$-complete.

  Assume a polynomial-time approximation algorithm $A$ for the BPP exists such that $OPT(I) > \frac{2}{3}\, A(I)$ for all instances $I$.

  Execute $A$ for an instance $\hat{I}$ of the BPP defined by $(w_1, \ldots, w_n)$ and $c = \sum_{j=1}^{n} w_j / 2$.

## Best polynomially achievable worst-case performance

- Worst-case of FFD and BFD: $\overline{r}(FFD) = \overline{r}(BFD) = \frac{3}{2}$.
- Can we find a better algorithm? **Bad news:**
- **No polynomial-time approximation algorithm for the BPP can have a WCPR smaller than $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$.**
- $\textsc{Partition}$ problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$?

  $\textsc{Partition}$ is $\mathcal{NP}$-complete.

  Assume a polynomial-time approximation algorithm $A$ for the BPP exists such that *OPT*$(I) > \frac{2}{3}$ *A*$(I)$ for all instances $I$.

  Execute $A$ for an instance $\hat{I}$ of the BPP defined by $(w_1, \ldots, w_n)$ and $c = \sum_{j=1}^{n} w_j/2$.

  **if** $A(\hat{I}) = 2$ **then** we know that the answer to $\textsc{Partition}$ is yes;

## Best polynomially achievable worst-case performance

- Worst-case of FFD and BFD: $\overline{r}(FFD) = \overline{r}(BFD) = \frac{3}{2}$.
- Can we find a better algorithm? **Bad news:**
- **No polynomial-time approximation algorithm for the BPP can have a WCPR smaller than $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$.**
- PARTITION problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$?

  PARTITION is $\mathcal{NP}$-complete.

  Assume a polynomial-time approximation algorithm $A$ for the BPP exists such that $OPT(I) > \frac{2}{3} A(I)$ for all instances $I$.

  Execute $A$ for an instance $\hat{I}$ of the BPP defined by $(w_1, \ldots, w_n)$ and $c = \sum_{j=1}^{n} w_j / 2$.

  **if** $A(\hat{I}) = 2$ **then** we know that the answer to PARTITION is yes;

  **else** $(A(\hat{I}) \geq 3)$ we know that $OPT(\hat{I}) > \frac{2}{3}3$, i.e., that $OPT(\hat{I}) > 2$,

# Best polynomially achievable worst-case performance

- Worst-case of FFD and BFD: $\overline{r}(FFD) = \overline{r}(BFD) = \frac{3}{2}$.
- Can we find a better algorithm? **Bad news:**
- **No polynomial-time approximation algorithm for the BPP can have a WCPR smaller than $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$.**
- Partition problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$?

  Partition is $\mathcal{NP}$-complete.

  Assume a polynomial-time approximation algorithm $A$ for the BPP exists such that $OPT(I) > \frac{2}{3} A(I)$ for all instances $I$.

  Execute $A$ for an instance $\hat{I}$ of the BPP defined by $(w_1, \ldots, w_n)$ and $c = \sum_{j=1}^{n} w_j / 2$.

  **if** $A(\hat{I}) = 2$ **then** we know that the answer to Partition is `yes`;

  **else** $(A(\hat{I}) \geq 3)$ we know that $OPT(\hat{I}) > \frac{2}{3} 3$, i.e., that $OPT(\hat{I}) > 2$,

  and hence the answer to Partition is `no`.

## Best polynomially achievable worst-case performance

- Worst-case of FFD and BFD: $\overline{r}(FFD) = \overline{r}(BFD) = \frac{3}{2}$.
- Can we find a better algorithm? **Bad news:**
- **No polynomial-time approximation algorithm for the BPP can have a WCPR smaller than $\frac{3}{2}$ unless $\mathcal{P} = \mathcal{NP}$.**
- PARTITION problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$?

  PARTITION is $\mathcal{NP}$-complete.

  Assume a polynomial-time approximation algorithm $A$ for the BPP exists such that *OPT*$(I) > \frac{2}{3}$ *A*$(I)$ for all instances $I$.

  Execute $A$ for an instance $\hat{I}$ of the BPP defined by $(w_1, \ldots, w_n)$ and $c = \sum_{j=1}^{n} w_j / 2$.

  **if** $A(\hat{I}) = 2$ **then** we know that the answer to PARTITION is `yes`;

  **else** $(A(\hat{I}) \geq 3)$ we know that *OPT*$(\hat{I}) > \frac{2}{3} 3$, i.e., that *OPT*$(\hat{I}) > 2$,

  and hence the answer to PARTITION is `no`.

  In other words, we could solve PARTITION in polynomial time! $\square$

# Asymptotic worst-case performance

## Asymptotic worst-case performance

- FFD and BFD provide the best possible WCPR $\Longrightarrow$ the study approximation algorithms focused on a different performance ratio.

## Asymptotic worst-case performance

- FFD and BFD provide the best possible WCPR $\Longrightarrow$ the study approximation algorithms focused on a different performance ratio.
- Already in the mid-Seventies D. Johnson proved that $FFD(I) \leq \dfrac{11}{9}\,OPT(I) + 4 \;\; \forall\, I$.

## Asymptotic worst-case performance

- FFD and BFD provide the best possible WCPR $\Longrightarrow$ the study approximation algorithms focused on a different performance ratio.
- Already in the mid-Seventies D. Johnson proved that $FFD(I) \leq \dfrac{11}{9}\,OPT(I) + 4 \;\; \forall\, I$.
- **Asymptotic worst-case performance ratio** of an approximation algorithm $A =$

  smallest real number $\overline{r}^{\infty}(A) > 1$ such that, for some positive integer $k$,
  $A(I)/OPT(I) \leq \overline{r}^{\infty}(A)$ for all instances $I$ satisfying $OPT(I) \geq k$.

## Asymptotic worst-case performance

- FFD and BFD provide the best possible WCPR $\Longrightarrow$ the study approximation algorithms focused on a different performance ratio.
- Already in the mid-Seventies D. Johnson proved that $FFD(I) \le \frac{11}{9}\, OPT(I) + 4 \;\; \forall\, I$.
- **Asymptotic worst-case performance ratio** of an approximation algorithm $A =$

  smallest real number $\overline{r}^{\infty}(A) > 1$ such that, for some positive integer $k$,
  $A(I)/OPT(I) \le \overline{r}^{\infty}(A)$ for all instances $I$ satisfying $OPT(I) \ge k$.
- $\overline{r}^{\infty}(FFD) = \overline{r}^{\infty}(BFD) = \frac{11}{9}$.

# Asymptotic worst-case performance

- FFD and BFD provide the best possible WCPR $\Longrightarrow$ the study approximation algorithms focused on a different performance ratio.
- Already in the mid-Seventies D. Johnson proved that $FFD(I) \leq \frac{11}{9} OPT(I) + 4 \;\; \forall \; I$.
- **Asymptotic worst-case performance ratio** of an approximation algorithm $A =$

  smallest real number $\overline{r}^{\infty}(A) > 1$ such that, for some positive integer $k$, $A(I)/OPT(I) \leq \overline{r}^{\infty}(A)$ for all instances $I$ satisfying $OPT(I) \geq k$.
- $\overline{r}^{\infty}(FFD) = \overline{r}^{\infty}(BFD) = \frac{11}{9}$.
- Impressive number of results, of mostly theoretical relevance (see surveys).

# Asymptotic worst-case performance

- FFD and BFD provide the best possible WCPR $\Longrightarrow$ the study approximation algorithms focused on a different performance ratio.
- Already in the mid-Seventies D. Johnson proved that $FFD(I) \leq \frac{11}{9}OPT(I) + 4 \ \ \forall\ I$.
- **Asymptotic worst-case performance ratio** of an approximation algorithm $A =$

  smallest real number $\overline{r}^{\infty}(A) > 1$ such that, for some positive integer $k$, $A(I)/OPT(I) \leq \overline{r}^{\infty}(A)$ for all instances $I$ satisfying $OPT(I) \geq k$.
- $\overline{r}^{\infty}(FFD) = \overline{r}^{\infty}(BFD) = \frac{11}{9}$.
- Impressive number of results, of mostly theoretical relevance (see surveys).
- "History" of the $11/9$ ratio:
  - Johnson (1974): $FFD(I) \leq \frac{11}{9}OPT(I)$ **+ 4** $\ \forall I$.

# Asymptotic worst-case performance

- FFD and BFD provide the best possible WCPR $\Longrightarrow$ the study approximation algorithms focused on a different performance ratio.
- Already in the mid-Seventies D. Johnson proved that $FFD(I) \leq \dfrac{11}{9} OPT(I) + 4 \;\; \forall\, I$.
- **Asymptotic worst-case performance ratio** of an approximation algorithm $A =$

  smallest real number $\overline{r}^{\infty}(A) > 1$ such that, for some positive integer $k$, $A(I)/OPT(I) \leq \overline{r}^{\infty}(A)$ for all instances $I$ satisfying $OPT(I) \geq k$.
- $\overline{r}^{\infty}(FFD) = \overline{r}^{\infty}(BFD) = \dfrac{11}{9}$.
- Impressive number of results, of mostly theoretical relevance (see surveys).
- "History" of the $11/9$ ratio:
  - Johnson (1974): $FFD(I) \leq \dfrac{11}{9} OPT(I)$ $\mathbf{+\ 4}$ $\forall I$. **Proof: 100 pages**;

# Asymptotic worst-case performance

- FFD and BFD provide the best possible WCPR $\Longrightarrow$ the study approximation algorithms focused on a different performance ratio.
- Already in the mid-Seventies D. Johnson proved that $FFD(I) \leq \frac{11}{9}OPT(I) + 4 \;\; \forall\, I$.
- **Asymptotic worst-case performance ratio** of an approximation algorithm $A =$

  smallest real number $\overline{r}^{\infty}(A) > 1$ such that, for some positive integer $k$, $A(I)/OPT(I) \leq \overline{r}^{\infty}(A)$ for all instances $I$ satisfying $OPT(I) \geq k$.
- $\overline{r}^{\infty}(FFD) = \overline{r}^{\infty}(BFD) = \frac{11}{9}$.
- Impressive number of results, of mostly theoretical relevance (see surveys).
- "History" of the $11/9$ ratio:
  - Johnson (1974): $FFD(I) \leq \frac{11}{9}OPT(I) + \mathbf{4} \;\; \forall I$. **Proof: 100 pages**;
  - Baker (1985): $FFD(I) \leq \frac{11}{9}OPT(I) + \mathbf{3} \;\; \forall I$.

## Asymptotic worst-case performance

- FFD and BFD provide the best possible WCPR $\Longrightarrow$ the study approximation algorithms focused on a different performance ratio.
- Already in the mid-Seventies D. Johnson proved that $FFD(I) \leq \frac{11}{9} OPT(I) + 4 \ \ \forall \ I$.
- **Asymptotic worst-case performance ratio** of an approximation algorithm $A =$

  smallest real number $\overline{r}^{\infty}(A) > 1$ such that, for some positive integer $k$, $A(I)/OPT(I) \leq \overline{r}^{\infty}(A)$ for all instances $I$ satisfying $OPT(I) \geq k$.
- $\overline{r}^{\infty}(FFD) = \overline{r}^{\infty}(BFD) = \frac{11}{9}$.
- Impressive number of results, of mostly theoretical relevance (see surveys).
- "History" of the $11/9$ ratio:
  - Johnson (1974): $FFD(I) \leq \frac{11}{9} OPT(I)$ **+ 4** $\forall I$. **Proof: 100 pages**;
  - Baker (1985): $FFD(I) \leq \frac{11}{9} OPT(I)$ **+ 3** $\forall I$. **Proof: 20 pages**;

## Asymptotic worst-case performance

- FFD and BFD provide the best possible WCPR $\Longrightarrow$ the study approximation algorithms focused on a different performance ratio.
- Already in the mid-Seventies D. Johnson proved that $FFD(I) \leq \frac{11}{9} OPT(I) + 4 \;\; \forall \, I$.
- **Asymptotic worst-case performance ratio** of an approximation algorithm $A =$

  smallest real number $\overline{r}^{\infty}(A) > 1$ such that, for some positive integer $k$, $A(I)/OPT(I) \leq \overline{r}^{\infty}(A)$ for all instances $I$ satisfying $OPT(I) \geq k$.
- $\overline{r}^{\infty}(FFD) = \overline{r}^{\infty}(BFD) = \frac{11}{9}$.
- Impressive number of results, of mostly theoretical relevance (see surveys).
- "History" of the $11/9$ ratio:
  - Johnson (1974): $FFD(I) \leq \frac{11}{9} OPT(I)$ **+ 4** $\forall I$. **Proof: 100 pages**;
  - Baker (1985): $FFD(I) \leq \frac{11}{9} OPT(I)$ **+ 3** $\forall I$. **Proof: 20 pages**;
  - Yue (1991): $FFD(I) \leq \frac{11}{9} OPT(I)$ **+ 1** $\forall I$.

# Asymptotic worst-case performance

- FFD and BFD provide the best possible WCPR $\Longrightarrow$ the study approximation algorithms focused on a different performance ratio.
- Already in the mid-Seventies D. Johnson proved that $FFD(I) \le \frac{11}{9} OPT(I) + 4 \;\; \forall\, I$.
- **Asymptotic worst-case performance ratio** of an approximation algorithm $A =$

  smallest real number $\overline{r}^{\infty}(A) > 1$ such that, for some positive integer $k$, $A(I)/OPT(I) \le \overline{r}^{\infty}(A)$ for all instances $I$ satisfying $OPT(I) \ge k$.
- $\overline{r}^{\infty}(FFD) = \overline{r}^{\infty}(BFD) = \frac{11}{9}$.
- Impressive number of results, of mostly theoretical relevance (see surveys).
- "History" of the $11/9$ ratio:
  - Johnson (1974): $FFD(I) \le \frac{11}{9} OPT(I)$ **+ 4** $\forall I$. **Proof: 100 pages**;
  - Baker (1985): $FFD(I) \le \frac{11}{9} OPT(I)$ **+ 3** $\forall I$. **Proof: 20 pages**;
  - Yue (1991): $FFD(I) \le \frac{11}{9} OPT(I)$ **+ 1** $\forall I$. **Proof: 10 pages**.

# Asymptotic worst-case of on-line algorithms

# Asymptotic worst-case of on-line algorithms

| Algorithm | Time | $\overline{r}^{\infty}(A)$ |
|---|---|---|
| NF | $O(n)$ | 2 |
| WF | $O(n \log n)$ | 2 |
| FF | $O(n \log n)$ | 1.7 |
| BF | $O(n \log n)$ | 1.7 |

# Asymptotic worst-case of on-line algorithms

| Algorithm | Time | $\overline{r}^{\infty}(A)$ |
|---|---|---|
| NF | $O(n)$ | 2 |
| WF | $O(n \log n)$ | 2 |
| FF | $O(n \log n)$ | 1.7 |
| BF | $O(n \log n)$ | 1.7 |

- **Any-Fit constraint:**

## Asymptotic worst-case of on-line algorithms

| Algorithm | Time | $\overline{r}^{\infty}(A)$ |
|---|---|---|
| NF | $O(n)$ | 2 |
| WF | $O(n \log n)$ | 2 |
| FF | $O(n \log n)$ | 1.7 |
| BF | $O(n \log n)$ | 1.7 |

- **Any-Fit constraint:**

  *If $B_1, \ldots, B_i$ are the current non-empty bins, then the current item*

  will not be packed into $B_{i+1}$ unless it does not fit in any of the bins $B_1, \ldots, B_i$.

# Asymptotic worst-case of on-line algorithms

| Algorithm | Time | $\overline{r}^\infty(A)$ |
|---|---|---|
| NF | $O(n)$ | 2 |
| WF | $O(n \log n)$ | 2 |
| FF | $O(n \log n)$ | 1.7 |
| BF | $O(n \log n)$ | 1.7 |

- **Any-Fit constraint:**

  *If $B_1, \dots, B_i$ are the current non-empty bins, then the current item*

  will not be packed into $B_{i+1}$ unless it does not fit in any of the bins $B_1, \dots, B_i$.
- $\mathcal{AF}$ = class of on-line heuristics satisfying the Any-Fit constraint.

# Asymptotic worst-case of on-line algorithms

| Algorithm | Time | $\overline{r}^{\infty}(A)$ |
|---|---|---|
| NF | $O(n)$ | 2 |
| WF | $O(n \log n)$ | 2 |
| FF | $O(n \log n)$ | 1.7 |
| BF | $O(n \log n)$ | 1.7 |

- **Any-Fit constraint:**

  *If $B_1, \dots, B_i$ are the current non-empty bins, then the current item*

  will not be packed into $B_{i+1}$ unless it does not fit in any of the bins $B_1, \dots, B_i$.
- $\mathcal{AF} =$ class of on-line heuristics satisfying the Any-Fit constraint.
- FF, WF, BF $\in \mathcal{AF}$.

## Asymptotic worst-case of on-line algorithms

| Algorithm | Time | $\overline{r}^{\infty}(A)$ |
|---|---|---|
| NF | $O(n)$ | 2 |
| WF | $O(n \log n)$ | 2 |
| FF | $O(n \log n)$ | 1.7 |
| BF | $O(n \log n)$ | 1.7 |

- **Any-Fit constraint:**

  *If $B_1, \ldots, B_i$ are the current non-empty bins, then the current item*

  will not be packed into $B_{i+1}$ unless it does not fit in any of the bins $B_1, \ldots, B_i$.

- $\mathcal{AF}$ = class of on-line heuristics satisfying the Any-Fit constraint.
- FF, WF, BF $\in \mathcal{AF}$.
- It can be proved that

  *For every algorithm $A \in \mathcal{AF}$,* $\quad \overline{r}^{\infty}(FF) \le \overline{r}^{\infty}(A) \le \overline{r}^{\infty}(WF)$

# Asymptotic worst-case of off-line algorithms

## Asymptotic worst-case of off-line algorithms

- *For any algorithm* $A \in \mathcal{AF}$ *that packs the items by nonincreasing size,*

$$\frac{11}{9} \;\le\; \overline{r}^{\infty}(A) \;\le\; \frac{5}{4}$$

## Asymptotic worst-case of off-line algorithms

- *For any algorithm $A \in \mathcal{AF}$ that packs the items by nonincreasing size,*

$$\frac{11}{9} \le \overline{r}^{\infty}(A) \le \frac{5}{4}$$

| Algorithm | Time | $\overline{r}^{\infty}(A)$ | |
|---|---|---|---|
| NFD | $O(n \log n)$ | 1.691... | Johnson et al., 1973-1974 |
| FFD | $O(n \log n)$ | 1.222... | Johnson et al., 1973-1974 |
| BFD | $O(n \log n)$ | 1.222... | Johnson et al., 1973-1974 |
| MFFD | $O(n \log n)$ | 1.183... | Garey & Johnson, 1985 |
| B2F | $O(n \log n)$ | 1.25 | Friesen & Langston, 1991 |
| CFB | $O(n \log n)$ | 1.16410... $\le \cdot \le$ 1.2 | Friesen & Langston, 1991 |
| GXFG | $O(n)$ | 1.5 | Johnson, 1974 |
| $H_4$ | $O(n)$ | 1.333... | Martel, 1985 |
| $H_7$ | $O(n)$ | 1.25 | Bekesi & Galambos, 1997 |

## Asymptotic worst-case of off-line algorithms

- *For any algorithm $A \in \mathcal{AF}$ that packs the items by nonincreasing size,*

$$\frac{11}{9} \le \overline{r}^{\infty}(A) \le \frac{5}{4}$$

| Algorithm | Time | $\overline{r}^{\infty}(A)$ | |
|---|---|---|---|
| NFD | $O(n \log n)$ | 1.691... | Johnson et al., 1973-1974 |
| FFD | $O(n \log n)$ | 1.222... | Johnson et al., 1973-1974 |
| BFD | $O(n \log n)$ | 1.222... | Johnson et al., 1973-1974 |
| MFFD | $O(n \log n)$ | 1.183... | Garey & Johnson, 1985 |
| B2F | $O(n \log n)$ | 1.25 | Friesen & Langston, 1991 |
| CFB | $O(n \log n)$ | 1.16410... $\le \cdot \le$ 1.2 | Friesen & Langston, 1991 |
| GXFG | $O(n)$ | 1.5 | Johnson, 1974 |
| $H_4$ | $O(n)$ | 1.333... | Martel, 1985 |
| $H_7$ | $O(n)$ | 1.25 | Bekesi & Galambos, 1997 |

**MFFD** (Modified FFD): Try to pack pairs of items with size in $(c/6, c/3]$ into bins containing a single item of size $> c/2$.

**B2F** (Best Two Fit): Fill one bin at a time, in greedy way; when no further item fits into the current bin, if the bin contains more than one item, try to replace the smallest item in the bin with a pair of unpacked items with size $\ge c/6$.

**CFB** (combined FFD–B2F): run both B2F and FFD and take the better packing.

# Approximation schemes

# Approximation schemes

- **Approximation scheme** = parametric family of approximation algorithms that produces a prefixed worst-case behavior.

# Approximation schemes

- **Approximation scheme** = parametric family of approximation algorithms that produces a prefixed worst-case behavior.
- **Question:** Does there exist an $\varepsilon > 0$ such that every $O(n)$-time algorithm $A$ must satisfy $\overline{r}^{\infty}(A) \geq 1 + \varepsilon$?

# Approximation schemes

- **Approximation scheme** = parametric family of approximation algorithms that produces a prefixed worst-case behavior.
- **Question:** Does there exist an $\varepsilon > 0$ such that every $O(n)$-time algorithm $A$ must satisfy $\overline{r}^{\infty}(A) \geq 1 + \varepsilon$?
- **Answer: No** (Fernandez de la Vega and Lueker, 1981):

  *For any $\varepsilon > 0$ there exists a linear-time algorithm $A_\varepsilon$ such that*

$$\overline{r}^{\infty}(A_\varepsilon) \leq 1 + \varepsilon \ \ \forall \varepsilon$$

# Approximation schemes

- **Approximation scheme** = parametric family of approximation algorithms that produces a prefixed worst-case behavior.
- **Question:** Does there exist an $\varepsilon > 0$ such that every $O(n)$-time algorithm $A$ must satisfy $\overline{r}^{\infty}(A) \geq 1 + \varepsilon$?
- **Answer: No** (Fernandez de la Vega and Lueker, 1981):

  *For any $\varepsilon > 0$ there exists a linear-time algorithm $A_\varepsilon$ such that*

$$\overline{r}^{\infty}(A_\varepsilon) \leq 1 + \varepsilon \;\; \forall \varepsilon$$

$A_\varepsilon$ is a **Polynomial-Time Approximation Scheme** based on:

- partitioning of the items (depending on $\varepsilon$);
- rounding techniques;
- solution of an LP relaxation;
- Next-Fit technique.

## Approximation schemes

- **Approximation scheme** = parametric family of approximation algorithms that produces a prefixed worst-case behavior.
- **Question:** Does there exist an $\varepsilon > 0$ such that every $O(n)$-time algorithm $A$ must satisfy $\overline{r}^{\infty}(A) \geq 1 + \varepsilon$?
- **Answer: No** (Fernandez de la Vega and Lueker, 1981):

  *For any $\varepsilon > 0$ there exists a linear-time algorithm $A_\varepsilon$ such that*

  $$\overline{r}^{\infty}(A_\varepsilon) \leq 1 + \varepsilon \;\; \forall \varepsilon$$

  $A_\varepsilon$ is a **Polynomial-Time Approximation Scheme** based on:

  - partitioning of the items (depending on $\varepsilon$);
  - rounding techniques;
  - solution of an LP relaxation;
  - Next-Fit technique.

- The time complexity of $A_\varepsilon$ is polynomial (linear) in $n$ $\forall$ $\varepsilon$

# Approximation schemes

- **Approximation scheme** $=$ parametric family of approximation algorithms that produces a prefixed worst-case behavior.
- **Question:** Does there exist an $\varepsilon > 0$ such that every $O(n)$-time algorithm $A$ must satisfy $\overline{r}^{\infty}(A) \geq 1 + \varepsilon$?
- **Answer: No** (Fernandez de la Vega and Lueker, 1981):

  *For any $\varepsilon > 0$ there exists a linear-time algorithm $A_\varepsilon$ such that*

$$\overline{r}^{\infty}(A_\varepsilon) \leq 1 + \varepsilon \ \ \forall \varepsilon$$

  $A_\varepsilon$ is a **Polynomial-Time Approximation Scheme** based on:

  - partitioning of the items (depending on $\varepsilon$);
  - rounding techniques;
  - solution of an LP relaxation;
  - Next-Fit technique.

- The time complexity of $A_\varepsilon$ is polynomial (linear) in $n$ $\forall\ \varepsilon$, but exponential in $\dfrac{1}{\varepsilon}$.

## Approximation schemes

- **Approximation scheme** $=$ parametric family of approximation algorithms that produces a prefixed worst-case behavior.

- **Question:** Does there exist an $\varepsilon > 0$ such that every $O(n)$-time algorithm $A$ must satisfy $\overline{r}^{\infty}(A) \geq 1 + \varepsilon$?

- **Answer: No** (Fernandez de la Vega and Lueker, 1981):

  *For any $\varepsilon > 0$ there exists a linear-time algorithm $A_\varepsilon$ such that*

$$\overline{r}^{\infty}(A_\varepsilon) \leq 1 + \varepsilon \;\; \forall \varepsilon$$

  $A_\varepsilon$ is a **Polynomial-Time Approximation Scheme** based on:

  - partitioning of the items (depending on $\varepsilon$);
  - rounding techniques;
  - solution of an LP relaxation;
  - Next-Fit technique.

- The time complexity of $A_\varepsilon$ is polynomial (linear) in $n$ $\forall$ $\varepsilon$, but exponential in $\frac{1}{\varepsilon}$.

- Improved by Karmarkar and Karp, 1982: **Fully Polynomial-Time Approximation Scheme**; time complexity polynomial (linear) both in $n$ and $\frac{1}{\varepsilon}$.

# Lower bounds

# Lower bounds

- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.

# Lower bounds

- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.
- In the optimal solution at most one bin can have a total contents $\leq \dfrac{c}{2}$.

## Lower bounds

- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.
- In the optimal solution at most one bin can have a total contents $\le \dfrac{c}{2}$.

$\Longrightarrow \sum_{j=1}^{n} w_j > \frac{OPT(I)-1}{2} c \Longrightarrow OPT(I) \le 2 \frac{\sum_{j=1}^{n} w_j}{c} \le 2L_1$

## Lower bounds

- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.
- In the optimal solution at most one bin can have a total contents $\leq \dfrac{c}{2}$.

  $\Longrightarrow \sum_{j=1}^{n} w_j > \frac{OPT(I)-1}{2} c \Longrightarrow OPT(I) \leq 2\frac{\sum_{j=1}^{n} w_j}{c} \leq 2L_1$

  $\Longrightarrow \underline{r}(L_1) = \frac{1}{2}$ (worst case: $(w) = (\frac{c}{2}+1, \frac{c}{2}+1, \ldots))$

## Lower bounds

- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.
- In the optimal solution at most one bin can have a total contents $\leq \dfrac{c}{2}$.

  $\Longrightarrow \sum_{j=1}^{n} w_j > \frac{OPT(I)-1}{2} c \Longrightarrow OPT(I) \leq 2 \frac{\sum_{j=1}^{n} w_j}{c} \leq 2L_1$

  $\Longrightarrow \underline{r}(L_1) = \frac{1}{2}$ (worst case: $(w) = (\frac{c}{2} + 1, \frac{c}{2} + 1, \ldots))$
- **A better bound** (Martello and Toth, 1990). Given any integer $\alpha$ $(0 \leq \alpha \leq c/2)$, let

## Lower bounds

- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.
- In the optimal solution at most one bin can have a total contents $\leq \dfrac{c}{2}$.

$\Longrightarrow \sum_{j=1}^{n} w_j > \frac{OPT(I)-1}{2} c \Longrightarrow OPT(I) \leq 2 \frac{\sum_{j=1}^{n} w_j}{c} \leq 2L_1$

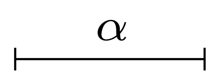
$\Longrightarrow \underline{r}(L_1) = \frac{1}{2}$ (worst case: $(w) = (\frac{c}{2}+1, \frac{c}{2}+1, \dots))$)

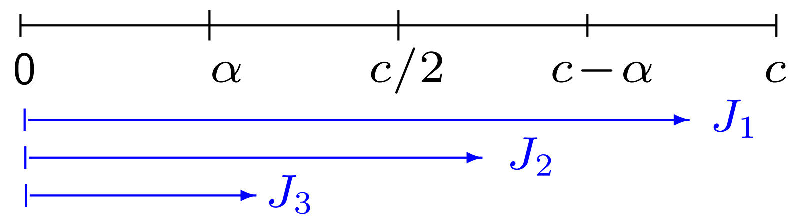
- **A better bound** (Martello and Toth, 1990). Given any integer $\alpha$ $(0 \leq \alpha \leq c/2)$, let

$J_1 = \{j \in N : w_j > c - \alpha\}$;

$J_2 = \{j \in N : c - \alpha \geq w_j > c/2\}$;

$J_3 = \{j \in N : c/2 \geq w_j \geq \alpha\}$,



# Lower bounds

- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.

- In the optimal solution at most one bin can have a total contents $\leq \dfrac{c}{2}$.

  $\Longrightarrow \sum_{j=1}^{n} w_j > \frac{OPT(I)-1}{2} c \Longrightarrow OPT(I) \leq 2\frac{\sum_{j=1}^{n} w_j}{c} \leq 2L_1$

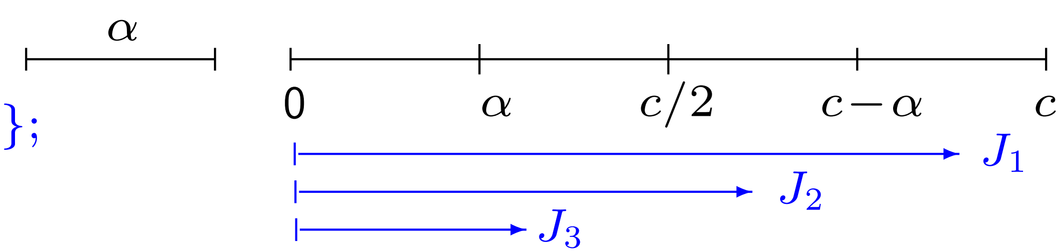
  $\Longrightarrow \underline{r}(L_1) = \frac{1}{2}$ (worst case: $(w) = (\frac{c}{2}+1, \frac{c}{2}+1, \ldots))$

- **A better bound** (Martello and Toth, 1990). Given any integer $\alpha$ $(0 \leq \alpha \leq c/2)$, let

  $J_1 = \{j \in N : w_j > c - \alpha\}$;

  $J_2 = \{j \in N : c - \alpha \geq w_j > c/2\}$;

  $J_3 = \{j \in N : c/2 \geq w_j \geq \alpha\}$,

  each item in $J_1 \cup J_2$ needs a separate bin,

# Lower bounds

- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.

- In the optimal solution at most one bin can have a total contents $\leq \dfrac{c}{2}$.

  $\Longrightarrow \sum_{j=1}^{n} w_j > \frac{OPT(I)-1}{2} c \Longrightarrow OPT(I) \leq 2 \frac{\sum_{j=1}^{n} w_j}{c} \leq 2L_1$

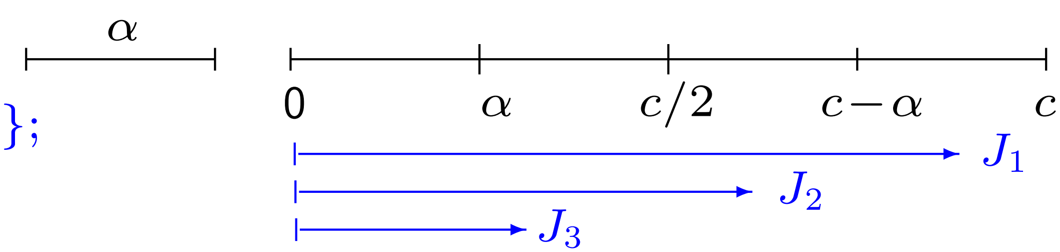
  $\Longrightarrow \underline{r}(L_1) = \frac{1}{2}$ (worst case: $(w) = (\frac{c}{2}+1, \frac{c}{2}+1, \ldots))$

- **A better bound** (Martello and Toth, 1990). Given any integer $\alpha$ $(0 \leq \alpha \leq c/2)$, let

  $J_1 = \{j \in N : w_j > c - \alpha\}$;

  $J_2 = \{j \in N : c - \alpha \geq w_j > c/2\}$;

  $J_3 = \{j \in N : c/2 \geq w_j \geq \alpha\}$,




  each item in $J_1 \cup J_2$ needs a separate bin,

  no item of $J_3$ can go to a bin containing an item of $J_1$. Then

## Lower bounds

- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.

- In the optimal solution at most one bin can have a total contents $\leq \dfrac{c}{2}$.

  $\Longrightarrow \sum_{j=1}^{n} w_j > \frac{OPT(I)-1}{2} c \Longrightarrow OPT(I) \leq 2\frac{\sum_{j=1}^{n} w_j}{c} \leq 2L_1$

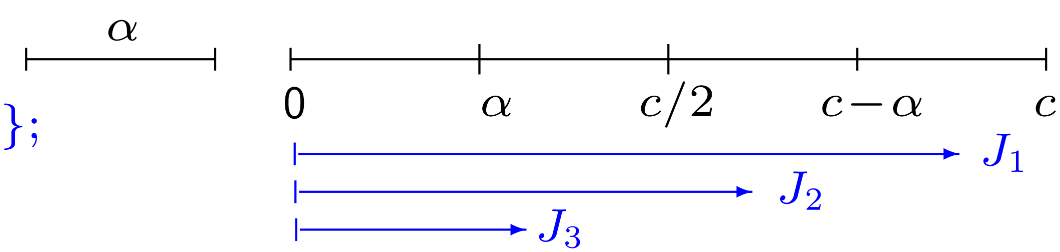
  $\Longrightarrow \underline{r}(L_1) = \frac{1}{2}$ (worst case: $(w) = (\frac{c}{2}+1, \frac{c}{2}+1, \ldots))$

- **A better bound** (Martello and Toth, 1990). Given any integer $\alpha$ $(0 \leq \alpha \leq c/2)$, let

  $J_1 = \{j \in N : w_j > c - \alpha\}$;

  $J_2 = \{j \in N : c - \alpha \geq w_j > c/2\}$;

  $J_3 = \{j \in N : c/2 \geq w_j \geq \alpha\}$,




  each item in $J_1 \cup J_2$ needs a separate bin,

  no item of $J_3$ can go to a bin containing an item of $J_1$. Then

  $L(\alpha) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{\sum_{j \in J_3} w_j - (|J_2|c - \sum_{j \in J_2} w_j)}{c} \right\rceil\right)$ is a valid lower bound.

# Lower bounds

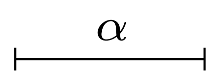
- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.
- In the optimal solution at most one bin can have a total contents $\leq \dfrac{c}{2}$.

$\Longrightarrow \sum_{j=1}^{n} w_j > \frac{OPT(I)-1}{2} c \Longrightarrow OPT(I) \leq 2 \frac{\sum_{j=1}^{n} w_j}{c} \leq 2L_1$

$\Longrightarrow \underline{r}(L_1) = \frac{1}{2}$ (worst case: $(w) = (\frac{c}{2}+1, \frac{c}{2}+1, \dots))$

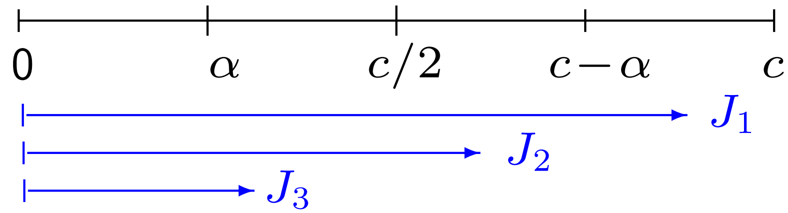
- **A better bound** (Martello and Toth, 1990). Given any integer $\alpha$ $(0 \leq \alpha \leq c/2)$, let

$J_1 = \{j \in N : w_j > c - \alpha\};$




$J_2 = \{j \in N : c - \alpha \geq w_j > c/2\};$

$J_3 = \{j \in N : c/2 \geq w_j \geq \alpha\},$




each item in $J_1 \cup J_2$ needs a separate bin,

no item of $J_3$ can go to a bin containing an item of $J_1$. Then

$L(\alpha) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{\sum_{j \in J_3} w_j - (|J_2| c - \sum_{j \in J_2} w_j)}{c} \right\rceil\right)$ is a valid lower bound.

The **overall bound** $L_2 = \max\{L(\alpha) : 0 \leq \alpha \leq c/2, \alpha \text{ integer}\}$

# Lower bounds

- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.
- In the optimal solution at most one bin can have a total contents $\leq \dfrac{c}{2}$.

$$\Longrightarrow \sum_{j=1}^{n} w_j > \frac{OPT(I)-1}{2} c \Longrightarrow OPT(I) \leq 2 \frac{\sum_{j=1}^{n} w_j}{c} \leq 2L_1$$

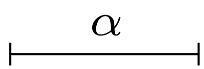
$\Longrightarrow \underline{r}(L_1) = \frac{1}{2}$ (worst case: $(w) = (\frac{c}{2}+1, \frac{c}{2}+1, \dots)$)

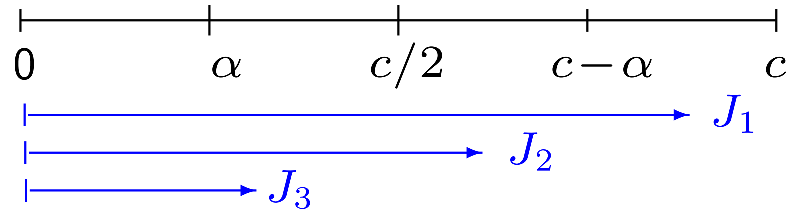
- **A better bound** (Martello and Toth, 1990). Given any integer $\alpha$ $(0 \leq \alpha \leq c/2)$, let

$J_1 = \{j \in N : w_j > c - \alpha\}$;

$J_2 = \{j \in N : c - \alpha \geq w_j > c/2\}$;

$J_3 = \{j \in N : c/2 \geq w_j \geq \alpha\}$,




each item in $J_1 \cup J_2$ needs a separate bin,

no item of $J_3$ can go to a bin containing an item of $J_1$. Then

$L(\alpha) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{\sum_{j \in J_3} w_j - (|J_2|c - \sum_{j \in J_2} w_j)}{c} \right\rceil\right)$ is a valid lower bound.

The **overall bound** $L_2 = \max\{L(\alpha) : 0 \leq \alpha \leq c/2, \alpha \text{ integer}\}$

(1) can be computed in $O(n \log n)$ time;

# Lower bounds

- **Continuous relaxation**: $L_1 = \left\lceil \sum_{j=1}^{n} w_j / c \right\rceil$. Computable in $O(n)$ time.

- In the optimal solution at most one bin can have a total contents $\leq \dfrac{c}{2}$.

  $\Longrightarrow \sum_{j=1}^{n} w_j > \frac{OPT(I)-1}{2} c \Longrightarrow OPT(I) \leq 2 \frac{\sum_{j=1}^{n} w_j}{c} \leq 2L_1$

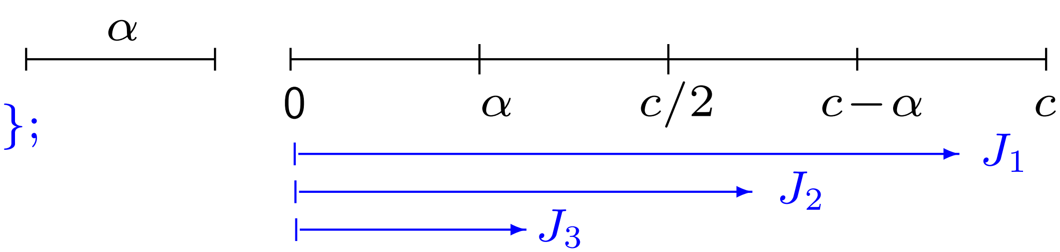
  $\Longrightarrow \underline{r}(L_1) = \frac{1}{2}$ (worst case: $(w) = (\frac{c}{2}+1, \frac{c}{2}+1, \dots))$

- **A better bound** (Martello and Toth, 1990). Given any integer $\alpha$ $(0 \leq \alpha \leq c/2)$, let

  $J_1 = \{j \in N : w_j > c - \alpha\}$;

  $J_2 = \{j \in N : c - \alpha \geq w_j > c/2\}$;

  $J_3 = \{j \in N : c/2 \geq w_j \geq \alpha\}$,




  each item in $J_1 \cup J_2$ needs a separate bin,

  no item of $J_3$ can go to a bin containing an item of $J_1$. Then

  $L(\alpha) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{\sum_{j \in J_3} w_j - (|J_2| c - \sum_{j \in J_2} w_j)}{c} \right\rceil\right)$ is a valid lower bound.

  The **overall bound** $L_2 = \max\{L(\alpha) : 0 \leq \alpha \leq c/2, \alpha \text{ integer}\}$

  (1) can be computed in $O(n \log n)$ time;

  (2) has **WCPR equal to** $\frac{2}{3}$.

## Best polynomially achievable worst-case performance

## Best polynomially achievable worst-case performance

- Worst-case of $L_2$: $\frac{2}{3}$.
- Can we find a better lower bound?

## Best polynomially achievable worst-case performance

- Worst-case of $L_2$: $\frac{2}{3}$.
- Can we find a better lower bound?
- **No lower bound, computable in polynomial time, for the BPP can have a WCPR greater than $\frac{2}{3}$ unless $\mathcal{P} = \mathcal{NP}$.**
- PARTITION problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$? ($\mathcal{NP}$-complete).

## Best polynomially achievable worst-case performance

- Worst-case of $L_2$: $\frac{2}{3}$.
- Can we find a better lower bound?
- **No lower bound, computable in polynomial time, for the BPP can have a WCPR greater than $\frac{2}{3}$ unless $\mathcal{P} = \mathcal{NP}$.**
- $\textsc{Partition}$ problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$? ($\mathcal{NP}$-complete).

  Assume a polynomial-time lower bound $L$ exists such that $OPT(I) < \frac{3}{2}\, L(I)\ \forall$ instances $I$.

## Best polynomially achievable worst-case performance

- Worst-case of $L_2$: $\frac{2}{3}$.
- Can we find a better lower bound?
- **No lower bound, computable in polynomial time, for the BPP can have a WCPR greater than $\frac{2}{3}$ unless $\mathcal{P} = \mathcal{NP}$.**
- PARTITION problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$? ($\mathcal{NP}$-complete).

  Assume a polynomial-time lower bound $L$ exists such that $OPT(I) < \frac{3}{2}\, L(I)\ \forall$ instances $I$.

  Compute $L$ for instance $\hat{I}$ of the BPP defined by $(w_1, \ldots, w_n)$ and $c = \sum_{j=1}^{n} w_j / 2$.

# Best polynomially achievable worst-case performance

- Worst-case of $L_2$: $\frac{2}{3}$.
- Can we find a better lower bound?
- **No lower bound, computable in polynomial time, for the BPP can have a WCPR greater than $\frac{2}{3}$ unless $\mathcal{P} = \mathcal{NP}$.**
- PARTITION problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$? ($\mathcal{NP}$-complete).

  Assume a polynomial-time lower bound $L$ exists such that $OPT(I) < \frac{3}{2}\, L(I)\ \forall$ instances $I$.

  Compute $L$ for instance $\hat{I}$ of the BPP defined by $(w_1, \ldots, w_n)$ and $c = \sum_{j=1}^{n} w_j / 2$.

  **if** $L(\hat{I}) \geq 3$ **then** we know that the answer to PARTITION is `no`.;

## Best polynomially achievable worst-case performance

- Worst-case of $L_2$: $\frac{2}{3}$.
- Can we find a better lower bound?
- **No lower bound, computable in polynomial time, for the BPP can have a WCPR greater than $\frac{2}{3}$ unless $\mathcal{P} = \mathcal{NP}$.**
- PARTITION problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$? ($\mathcal{NP}$-complete).

  Assume a polynomial-time lower bound $L$ exists such that $OPT(I) < \frac{3}{2}\, L(I)$ $\forall$ instances $I$.

  Compute $L$ for instance $\hat{I}$ of the BPP defined by $(w_1, \ldots, w_n)$ and $c = \sum_{j=1}^{n} w_j / 2$.

  **if** $L(\hat{I}) \geq 3$ **then** we know that the answer to PARTITION is `no`.;

  **else** $(L(\hat{I}) = 2)$ we know that $OPT(\hat{I}) < \frac{3}{2}\, 2$, i.e., $OPT(\hat{I}) = 2$ and the answer to PARTITION is `yes`.

## Best polynomially achievable worst-case performance

- Worst-case of $L_2$: $\frac{2}{3}$.
- Can we find a better lower bound?
- **No lower bound, computable in polynomial time, for the BPP can have a WCPR greater than $\frac{2}{3}$ unless $\mathcal{P} = \mathcal{NP}$.**
- PARTITION problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$? ($\mathcal{NP}$-complete).

  Assume a polynomial-time lower bound $L$ exists such that $OPT(I) < \frac{3}{2}\, L(I)\ \forall$ instances $I$.

  Compute $L$ for instance $\hat{I}$ of the BPP defined by $(w_1, \ldots, w_n)$ and $c = \sum_{j=1}^{n} w_j / 2$.

  **if** $L(\hat{I}) \geq 3$ **then** we know that the answer to PARTITION is `no`.;

  **else** $(L(\hat{I}) = 2)$ we know that $OPT(\hat{I}) < \frac{3}{2}\, 2$, i.e., $OPT(\hat{I}) = 2$ and the answer to PARTITION is `yes`.

  In other words, we could solve PARTITION in polynomial time! $\square$

## Best polynomially achievable worst-case performance

- Worst-case of $L_2$: $\frac{2}{3}$.
- Can we find a better lower bound?
- **No lower bound, computable in polynomial time, for the BPP can have a WCPR greater than $\frac{2}{3}$ unless $\mathcal{P} = \mathcal{NP}$.**
- $\textsc{Partition}$ problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$? ($\mathcal{NP}$-complete).

  Assume a polynomial-time lower bound $L$ exists such that $OPT(I) < \frac{3}{2}\, L(I)\ \forall$ instances $I$.

  Compute $L$ for instance $\hat{I}$ of the BPP defined by $(w_1, \ldots, w_n)$ and $c = \sum_{j=1}^{n} w_j/2$.

  **if** $L(\hat{I}) \geq 3$ **then** we know that the answer to $\textsc{Partition}$ is `no`.;

  **else** $(L(\hat{I}) = 2)$ we know that $OPT(\hat{I}) < \frac{3}{2}2$, i.e., $OPT(\hat{I}) = 2$ and the answer to $\textsc{Partition}$ is `yes`.

  In other words, we could solve $\textsc{Partition}$ in polynomial time! $\square$
- Other lower bounds can have better practical performance (Labbé et al., Martello and Toth) and have asymptotic WCPR equal to $\frac{3}{4}$.

# Best polynomially achievable worst-case performance

- Worst-case of $L_2$: $\frac{2}{3}$.
- Can we find a better lower bound?
- **No lower bound, computable in polynomial time, for the BPP can have a WCPR greater than $\frac{2}{3}$ unless $\mathcal{P} = \mathcal{NP}$.**
- Partition problem: is it possible to partition $S = \{w_1, \ldots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$? ($\mathcal{NP}$-complete).

  Assume a polynomial-time lower bound $L$ exists such that $OPT(I) < \frac{3}{2}\, L(I)\ \forall$ instances $I$.

  Compute $L$ for instance $\hat{I}$ of the BPP defined by $(w_1, \ldots, w_n)$ and $c = \sum_{j=1}^{n} w_j/2$.

  **if** $L(\hat{I}) \geq 3$ **then** we know that the answer to Partition is no.;

  **else** $(L(\hat{I}) = 2)$ we know that $OPT(\hat{I}) < \frac{3}{2}\,2$, i.e., $OPT(\hat{I}) = 2$ and the answer to Partition is yes.

  In other words, we could solve Partition in polynomial time! $\square$
- Other lower bounds can have better practical performance (Labbé et al., Martello and Toth) and have asymptotic WCPR equal to $\frac{3}{4}$.
- Different types of lower bound computations are based on *dual feasible functions* (Lueker, Fekete and Schepers).

## Best polynomially achievable worst-case performance

- Worst-case of $L_2$: $\frac{2}{3}$.
- Can we find a better lower bound?
- **No lower bound, computable in polynomial time, for the BPP can have a WCPR greater than $\frac{2}{3}$ unless $\mathcal{P} = \mathcal{NP}$.**
- $\textsc{Partition}$ problem: is it possible to partition $S = \{w_1, \dots, w_n\}$ into $S_1$, $S_2$ so that $\sum_{j \in S_1} w_j = \sum_{j \in S_2} w_j$? ($\mathcal{NP}$-complete).

  Assume a polynomial-time lower bound $L$ exists such that $OPT(I) < \frac{3}{2}\, L(I)$ $\forall$ instances $I$.

  Compute $L$ for instance $\hat{I}$ of the BPP defined by $(w_1, \dots, w_n)$ and $c = \sum_{j=1}^{n} w_j/2$.

  **if** $L(\hat{I}) \geq 3$ **then** we know that the answer to $\textsc{Partition}$ is `no`.;

  **else** $(L(\hat{I}) = 2)$ we know that $OPT(\hat{I}) < \frac{3}{2}2$, i.e., $OPT(\hat{I}) = 2$ and the answer to $\textsc{Partition}$ is `yes`.

  In other words, we could solve $\textsc{Partition}$ in polynomial time! $\Box$
- Other lower bounds can have better practical performance (Labbé et al., Martello and Toth) and have asymptotic WCPR equal to $\frac{3}{4}$.
- Different types of lower bound computations are based on *dual feasible functions* (Lueker, Fekete and Schepers).
- Methods to improve on a lower bound value (Dell'Amico and Martello, Alvim et al., Haouari and Gharbi, Jarboui et al.)

# Reduction Algorithms

# Reduction Algorithms

- **Reduction Algorithm** = preprocessing procedure used to determine the optimal value of a subset of variables.

# Reduction Algorithms

- **Reduction Algorithm** $=$ preprocessing procedure used to determine the optimal value of a subset of variables.
- Numerical Example: $n = 12$, $c = 100$,
  $(w_j) =$ (99 93 90 88 80 10 10 6 5 5 4 4 ).

# Reduction Algorithms

- **Reduction Algorithm** = preprocessing procedure used to determine the optimal value of a subset of variables.
- Numerical Example: $n = 12$, $c = 100$,
  $(w_j) =$ (99 93 90 88 80 10 10 6 5 5 4 4 ).

  99 alone in a bin;

# Reduction Algorithms

- **Reduction Algorithm** $=$ preprocessing procedure used to determine the optimal value of a subset of variables.
- Numerical Example: $n = 12$, $c = 100$,
  $(w_j) =$ (99 93 90 88 80 10 10 6 5 5 4 4 ).

  99 alone in a bin;

  93 can be packed with at most one more item $\rightarrow$ packing it with 6 is dominating (largest item);

# Reduction Algorithms

- **Reduction Algorithm** $=$ preprocessing procedure used to determine the optimal value of a subset of variables.
- Numerical Example: $n = 12$, $c = 100$,
  $(w_j) =$ (99 93 90 88 80 10 10 6 5 5 4 4 ).

  99 alone in a bin;

  93 can be packed with at most one more item $\rightarrow$ packing it with 6 is dominating (largest item);

  reduced instance: $(w_j) =$ ( 90 88 80 10 10 5 5 4 4 );

# Reduction Algorithms

- **Reduction Algorithm** = preprocessing procedure used to determine the optimal value of a subset of variables.

- Numerical Example: $n = 12$, $c = 100$,
  $(w_j) =$ (99 93 90 88 80 10 10 6 5 5 4 4 ).

  99 alone in a bin;

  93 can be packed with at most one more item $\rightarrow$ packing it with 6 is dominating (largest item);

  reduced instance: $(w_j) =$ ( 90 88 80 10 10 5 5 4 4 );

  90 can be packed with at most two more items $\rightarrow$ packing it with 10 is dominating (bin full);

# Reduction Algorithms

- **Reduction Algorithm** = preprocessing procedure used to determine the optimal value of a subset of variables.
- Numerical Example: $n = 12$, $c = 100$,
  $(w_j) =$ (99 93 90 88 80 10 10 6 5 5 4 4 ).

  99 alone in a bin;

  93 can be packed with at most one more item $\rightarrow$ packing it with 6 is dominating (largest item);

  reduced instance: $(w_j) =$ ( 90 88 80 10 10 5 5 4 4 );

  90 can be packed with at most two more items $\rightarrow$ packing it with 10 is dominating (bin full);

  reduced instance: $(w_j) =$ ( 88 80 10 5 5 4 4 );

# Reduction Algorithms

- **Reduction Algorithm** = preprocessing procedure used to determine the optimal value of a subset of variables.
- Numerical Example: $n = 12$, $c = 100$,
  $(w_j) =$ (99 93 90 88 80 10 10 6 5 5 4 4 ).

  99 alone in a bin;

  93 can be packed with at most one more item $\longrightarrow$ packing it with 6 is dominating (largest item);

  reduced instance: $(w_j) =$ ( 90 88 80 10 10 5 5 4 4 );

  90 can be packed with at most two more items $\longrightarrow$ packing it with 10 is dominating (bin full);

  reduced instance: $(w_j) =$ ( 88 80 10 5 5 4 4 );

  88 can be packed with at most two more items $\longrightarrow$ packing it with 10 is dominating (10 $\geq$ maximum pair);

# Reduction Algorithms

- **Reduction Algorithm** = preprocessing procedure used to determine the optimal value of a subset of variables.
- Numerical Example: $n = 12$, $c = 100$,
  $(w_j) =$ (99 93 90 88 80 10 10 6 5 5 4 4 ).

  99 alone in a bin;

  93 can be packed with at most one more item $\longrightarrow$ packing it with 6 is dominating (largest item);

  reduced instance: $(w_j) =$ ( 90 88 80 10 10 5 5 4 4 );

  90 can be packed with at most two more items $\longrightarrow$ packing it with 10 is dominating (bin full);

  reduced instance: $(w_j) =$ ( 88 80 10 5 5 4 4 );

  88 can be packed with at most two more items $\longrightarrow$ packing it with 10 is dominating (10 $\geq$ maximum pair);

  reduced instance: $(w_j) =$ ( 80 5 5 4 4 ): one bin (optimal solution).

# Reduction Algorithms

- **Reduction Algorithm** = preprocessing procedure used to determine the optimal value of a subset of variables.
- Numerical Example: $n = 12$, $c = 100$,
  $(w_j) =$ (99 93 90 88 80 10 10 6 5 5 4 4 ).

  99 alone in a bin;

  93 can be packed with at most one more item $\rightarrow$ packing it with 6 is dominating (largest item);

  reduced instance: $(w_j) =$ ( 90 88 80 10 10 5 5 4 4 );

  90 can be packed with at most two more items $\rightarrow$ packing it with 10 is dominating (bin full);

  reduced instance: $(w_j) =$ ( 88 80 10 5 5 4 4 );

  88 can be packed with at most two more items $\rightarrow$ packing it with 10 is dominating (10 $\geq$ maximum pair);

  reduced instance: $(w_j) =$ ( 80 5 5 4 4 ): one bin (optimal solution).
- Ideas generalized to a
  general **Dominance Criterion** between pairs of subsets of items (Martello and Toth)

# Exact Algorithms: Branch-and-Bound

## Exact Algorithms: Branch-and-Bound

- Eilon and Christofides, 1971 (enumerative algorithm);
  Hung and Brown, 1978 (branch-and-bound);
  Martello and Toth, 1990 (specifically tailored branch-and-bound: MTP, popular Fortran code);
  Scholl, Klein and Jurgens, 1997 (BISON: MTP + Tabu search, Pascal code).

# Exact Algorithms: Branch-and-Bound

- Eilon and Christofides, 1971 (enumerative algorithm);
  Hung and Brown, 1978 (branch-and-bound);
  Martello and Toth, 1990 (specifically tailored branch-and-bound: MTP, popular Fortran code);
  Scholl, Klein and Jurgens, 1997 (BISON: MTP + Tabu search, Pascal code).
- Outline (MTP)
  - depth-first strategy;
  - items sorted by non-increasing size;

## Exact Algorithms: Branch-and-Bound

- Eilon and Christofides, 1971 (enumerative algorithm);
  Hung and Brown, 1978 (branch-and-bound);
  Martello and Toth, 1990 (specifically tailored branch-and-bound: MTP, popular Fortran code);
  Scholl, Klein and Jurgens, 1997 (BISON: MTP + Tabu search, Pascal code).
- Outline (MTP)
  - depth-first strategy;
  - items sorted by non-increasing size;
  - at each decision node, the first (largest) free item is assigned
    - to all feasible initialized bins,

# Exact Algorithms: Branch-and-Bound

- Eilon and Christofides, 1971 (enumerative algorithm);
  Hung and Brown, 1978 (branch-and-bound);
  Martello and Toth, 1990 (specifically tailored branch-and-bound: MTP, popular Fortran code);
  Scholl, Klein and Jurgens, 1997 (BISON: MTP + Tabu search, Pascal code).
- Outline (MTP)
  - depth-first strategy;
  - items sorted by non-increasing size;
  - at each decision node, the first (largest) free item is assigned
    * to all feasible initialized bins,
    * and, possibly, to a new bin.

## Exact Algorithms: Branch-and-Bound

- Eilon and Christofides, 1971 (enumerative algorithm);
  Hung and Brown, 1978 (branch-and-bound);
  Martello and Toth, 1990 (specifically tailored branch-and-bound: MTP, popular Fortran code);
  Scholl, Klein and Jurgens, 1997 (BISON: MTP + Tabu search, Pascal code).
- Outline (MTP)
  - depth-first strategy;
  - items sorted by non-increasing size;
  - at each decision node, the first (largest) free item is assigned
    * to all feasible initialized bins,
    * and, possibly, to a new bin.
  - At any *forward step*
    * lower bound computations ($L_2$ and $L_3$ (improved bound));
    * reduction of the current instance;

# Exact Algorithms: Branch-and-Bound

- Eilon and Christofides, 1971 (enumerative algorithm);
  Hung and Brown, 1978 (branch-and-bound);
  Martello and Toth, 1990 (specifically tailored branch-and-bound: MTP, popular Fortran code);
  Scholl, Klein and Jurgens, 1997 (BISON: MTP + Tabu search, Pascal code).
- Outline (MTP)
  - depth-first strategy;
  - items sorted by non-increasing size;
  - at each decision node, the first (largest) free item is assigned
    - to all feasible initialized bins,
    - and, possibly, to a new bin.
  - At any *forward step*
    - lower bound computations ($L_2$ and $L_3$ (improved bound));
    - reduction of the current instance;
    - if the node is not fathomed, FFD, BFD and WFD executed on the current problem to try and improve the incumbent solution.

# Exact Algorithms: Branch-and-Bound

- Eilon and Christofides, 1971 (enumerative algorithm);
  Hung and Brown, 1978 (branch-and-bound);
  Martello and Toth, 1990 (specifically tailored branch-and-bound: MTP, popular Fortran code);
  Scholl, Klein and Jurgens, 1997 (BISON: MTP + Tabu search, Pascal code).
- Outline (MTP)
  - depth-first strategy;
  - items sorted by non-increasing size;
  - at each decision node, the first (largest) free item is assigned
    - to all feasible initialized bins,
    - and, possibly, to a new bin.
  - At any *forward step*
    - lower bound computations ($L_2$ and $L_3$ (improved bound));
    - reduction of the current instance;
    - if the node is not fathomed, FFD, BFD and WFD executed on the current problem to try and improve the incumbent solution.
  - Dominance criterion between decision nodes.

# Exact Algorithms: Branch-and-Bound

- Eilon and Christofides, 1971 (enumerative algorithm);
  Hung and Brown, 1978 (branch-and-bound);
  Martello and Toth, 1990 (specifically tailored branch-and-bound: MTP, popular Fortran code);
  Scholl, Klein and Jurgens, 1997 (BISON: MTP + Tabu search, Pascal code).
- Outline (MTP)
  - depth-first strategy;
  - items sorted by non-increasing size;
  - at each decision node, the first (largest) free item is assigned
    - to all feasible initialized bins,
    - and, possibly, to a new bin.
  - At any *forward step*
    - lower bound computations ($L_2$ and $L_3$ (improved bound));
    - reduction of the current instance;
    - if the node is not fathomed, FFD, BFD and WFD executed on the current problem to try and improve the incumbent solution.
  - Dominance criterion between decision nodes.
  - Computations at the decision nodes:
    - for each initialized bin, create a "super item" having size = sum of the sizes of the items in the bin;

# Exact Algorithms: Branch-and-Bound

- Eilon and Christofides, 1971 (enumerative algorithm);
  Hung and Brown, 1978 (branch-and-bound);
  Martello and Toth, 1990 (specifically tailored branch-and-bound: MTP, popular Fortran code);
  Scholl, Klein and Jurgens, 1997 (BISON: MTP + Tabu search, Pascal code).
- Outline (MTP)
  - depth-first strategy;
  - items sorted by non-increasing size;
  - at each decision node, the first (largest) free item is assigned
    - to all feasible initialized bins,
    - and, possibly, to a new bin.
  - At any *forward step*
    - lower bound computations ($L_2$ and $L_3$ (improved bound));
    - reduction of the current instance;
    - if the node is not fathomed, FFD, BFD and WFD executed on the current problem to try and improve the incumbent solution.
  - Dominance criterion between decision nodes.
  - Computations at the decision nodes:
    - for each initialized bin, create a "super item" having size = sum of the sizes of the items in the bin;
    - lower bounds and reduction for the instance given by {super items} ∪ {free items}.

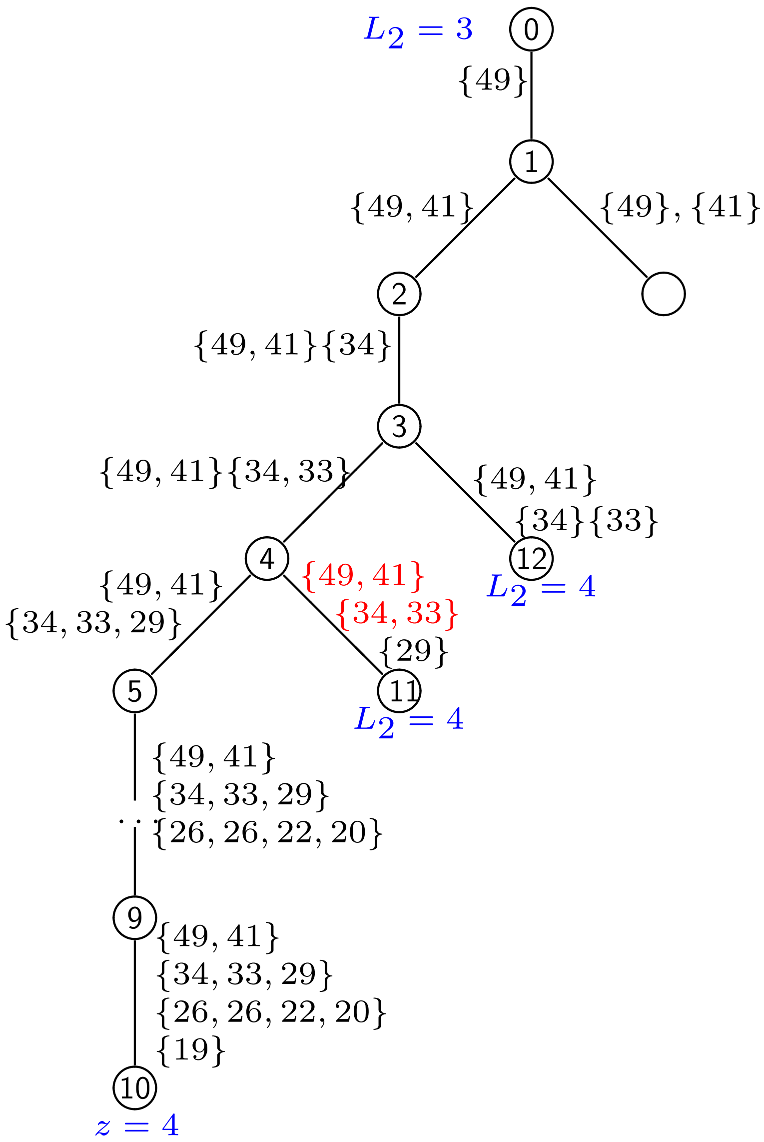
# Example

## Example

$n = 10$, $c = 100$, $(w_j) = ($49 41 34 33 29 26 26 22 20 19 ):

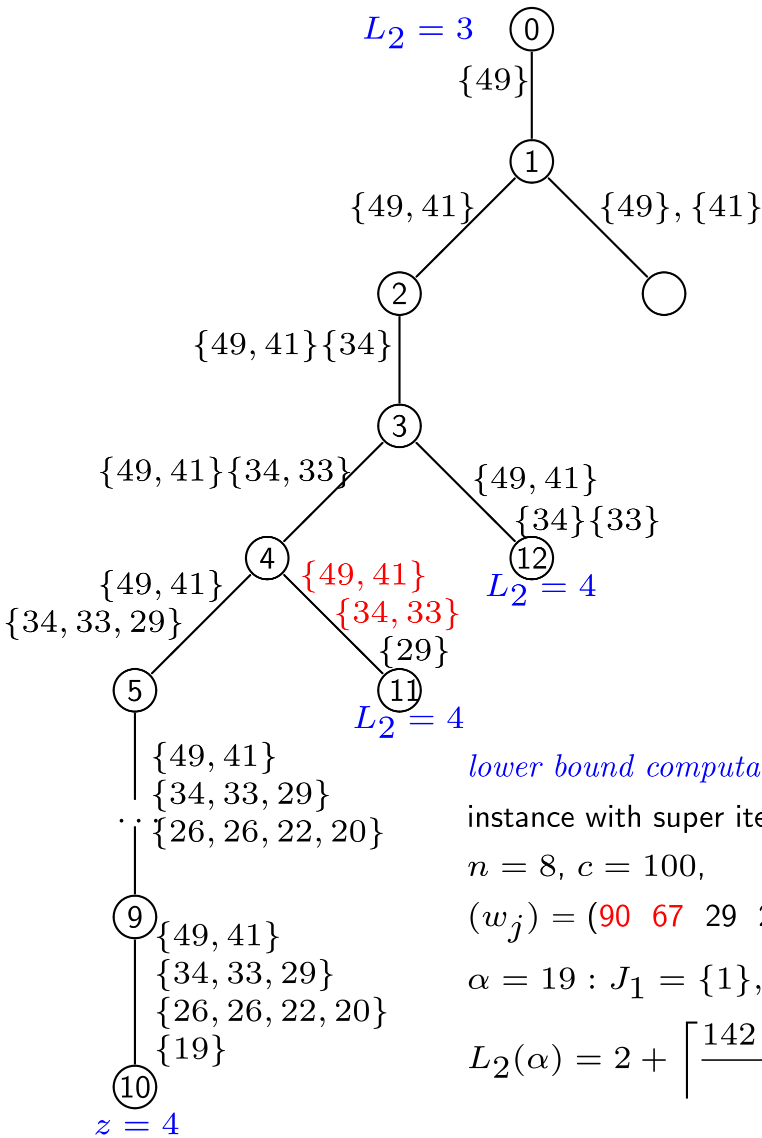
# Example

$n = 10$, $c = 100$, $(w_j) = (49 \quad 41 \quad 34 \quad 33 \quad 29 \quad 26 \quad 26 \quad 22 \quad 20 \quad 19\ )$:



# Example

$n = 10$, $c = 100$, $(w_j) = (49 \quad 41 \quad 34 \quad 33 \quad 29 \quad 26 \quad 26 \quad 22 \quad 20 \quad 19\ )$:




*lower bound computation at node 11*:

instance with super items:

$n = 8$, $c = 100$,

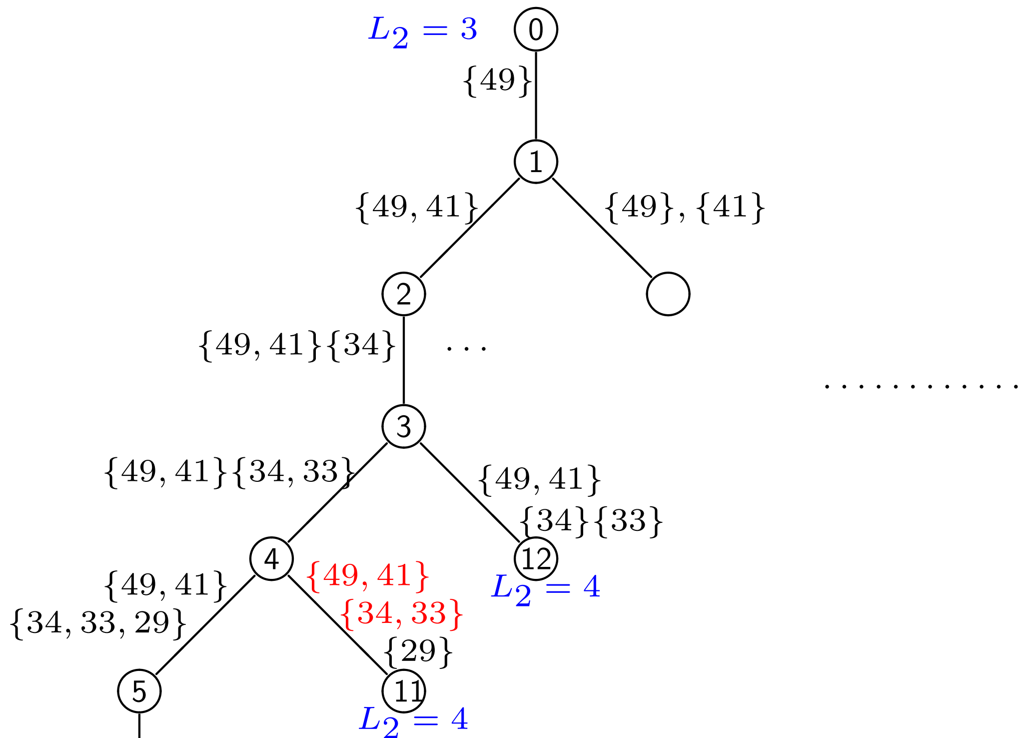
$(w_j) = (90 \;\; 67 \;\; 29 \;\; 26 \;\; 26 \;\; 22 \;\; 20 \;\; 19\ )$:

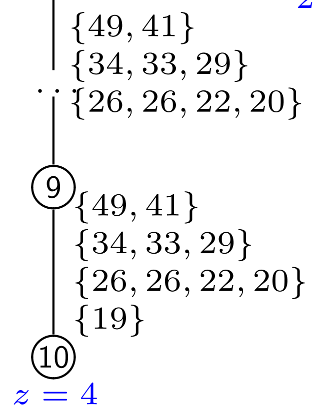
$\alpha = 19 : J_1 = \{1\}, J_2 = \{2\}, J_3 = \{3, \ldots, 8\};$

$$L_2(\alpha) = 2 + \left\lceil \frac{142 - (1 \cdot 100 - 67)}{100} \right\rceil = 4$$

# Example

$n = 10$, $c = 100$, $(w_j) = ($49 41 34 33 29 26 26 22 20 19 ):




*lower bound computation at node 11*:

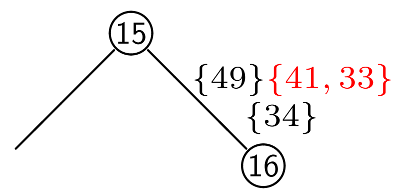
instance with super items:

$n = 8$, $c = 100$,

$(w_j) = ($90 67 29 26 26 22 20 19 ):

$\alpha = 19 : J_1 = \{1\}, J_2 = \{2\}, J_3 = \{3, \dots, 8\};$

$$L_2(\alpha) = 2 + \left\lceil \frac{142 - (1 \cdot 100 - 67)}{100} \right\rceil = 4$$




Reduction (super item 41+33=74):
$\{41, 33, 26\}$ removed from the instance
FFD on the residual instance:
$\{49, 29, 22\}$
$\{34, 26, 20, 19\}$
optimal

# Exact Algorithms: Branch-and-Bound/-and-Price/-and-Cut

## Exact Algorithms: Branch-and-Bound/-and-Price/-and-Cut

- Classical **Branch-and-Bound** for ILP/MILP: At each node of the branch-decision tree:
  - solve the LP relaxation of the (sub-)problem associated with the the current decision node;

# Exact Algorithms: Branch-and-Bound/-and-Price/-and-Cut

- Classical **Branch-and-Bound** for ILP/MILP: At each node of the branch-decision tree:
  - solve the LP relaxation of the (sub-)problem associated with the the current decision node;
  - if the solution is not integer, separate a fractional variable to get
    2 new sub-problems (2 new decision nodes);
  - continue until all decision nodes have been explored.

# Exact Algorithms: Branch-and-Bound/-and-Price/-and-Cut

- Classical **Branch-and-Bound** for ILP/MILP: At each node of the branch-decision tree:
  - solve the LP relaxation of the (sub-)problem associated with the the current decision node;
  - if the solution is not integer, separate a fractional variable to get 2 new sub-problems (2 new decision nodes);
  - continue until all decision nodes have been explored.
- **Branch-and-Cut:** ...(Branch-and-Bound) but
  - if the solution is not integer, before separating, add cutting planes to strengthen the relaxation, possibly finding an integer solution or improving on the lower bound value.

# Exact Algorithms: Branch-and-Bound/-and-Price/-and-Cut

- Classical **Branch-and-Bound** for ILP/MILP: At each node of the branch-decision tree:
  - solve the LP relaxation of the (sub-)problem associated with the the current decision node;
  - if the solution is not integer, separate a fractional variable to get 2 new sub-problems (2 new decision nodes);
  - continue until all decision nodes have been explored.
- **Branch-and-Cut:** ...(Branch-and-Bound) but
  - if the solution is not integer, before separating, add cutting planes to strengthen the relaxation, possibly finding an integer solution or improving on the lower bound value.
  - if the solution remains not integer, separate.

# Exact Algorithms: Branch-and-Bound/-and-Price/-and-Cut

- Classical **Branch-and-Bound** for ILP/MILP: At each node of the branch-decision tree:
  - solve the LP relaxation of the (sub-)problem associated with the the current decision node;
  - if the solution is not integer, separate a fractional variable to get 2 new sub-problems (2 new decision nodes);
  - continue until all decision nodes have been explored.
- **Branch-and-Cut:** ...(Branch-and-Bound) but
  - if the solution is not integer, before separating, add cutting planes to strengthen the relaxation, possibly finding an integer solution or improving on the lower bound value.
  - if the solution remains not integer, separate.
- **Branch-and-Price:** ...(Branch-and-Bound) but
  - use *column generation* to solve the LP relaxation at each node:

## Exact Algorithms: Branch-and-Bound/-and-Price/-and-Cut

- Classical **Branch-and-Bound** for ILP/MILP: At each node of the branch-decision tree:
  - solve the LP relaxation of the (sub-)problem associated with the the current decision node;
  - if the solution is not integer, separate a fractional variable to get 2 new sub-problems (2 new decision nodes);
  - continue until all decision nodes have been explored.
- **Branch-and-Cut:** ...(Branch-and-Bound) but
  - if the solution is not integer, before separating, add cutting planes to strengthen the relaxation, possibly finding an integer solution or improving on the lower bound value.
  - if the solution remains not integer, separate.
- **Branch-and-Price:** ...(Branch-and-Bound) but
  - use *column generation* to solve the LP relaxation at each node:
  - initially, only a subset of columns is included in the LP relaxation (*Restricted Master Problem*);

# Exact Algorithms: Branch-and-Bound/-and-Price/-and-Cut

- Classical **Branch-and-Bound** for ILP/MILP: At each node of the branch-decision tree:
  - solve the LP relaxation of the (sub-)problem associated with the the current decision node;
  - if the solution is not integer, separate a fractional variable to get 2 new sub-problems (2 new decision nodes);
  - continue until all decision nodes have been explored.
- **Branch-and-Cut:** ...(Branch-and-Bound) but
  - if the solution is not integer, before separating, add cutting planes to strengthen the relaxation, possibly finding an integer solution or improving on the lower bound value.
  - if the solution remains not integer, separate.
- **Branch-and-Price:** ...(Branch-and-Bound) but
  - use *column generation* to solve the LP relaxation at each node:
  - initially, only a subset of columns is included in the LP relaxation (*Restricted Master Problem*);
  - an auxiliary problem (*Pricing Problem*) is used to check optimality and to find columns to be added to improve the LP solution value;

# Exact Algorithms: Branch-and-Bound/-and-Price/-and-Cut

- Classical **Branch-and-Bound** for ILP/MILP: At each node of the branch-decision tree:
  - solve the LP relaxation of the (sub-)problem associated with the the current decision node;
  - if the solution is not integer, separate a fractional variable to get 2 new sub-problems (2 new decision nodes);
  - continue until all decision nodes have been explored.
- **Branch-and-Cut:** ...(Branch-and-Bound) but
  - if the solution is not integer, before separating, add cutting planes to strengthen the relaxation, possibly finding an integer solution or improving on the lower bound value.
  - if the solution remains not integer, separate.
- **Branch-and-Price:** ...(Branch-and-Bound) but
  - use *column generation* to solve the LP relaxation at each node:
  - initially, only a subset of columns is included in the LP relaxation (*Restricted Master Problem*);
  - an auxiliary problem (*Pricing Problem*) is used to check optimality and to find columns to be added to improve the LP solution value;
- **Branch-and-Price-and-Cut** = Branch-and-Bound + column generation + cutting planes.

# Exact Algorithms: Branch-and-Bound/-and-Price/-and-Cut

- Classical **Branch-and-Bound** for ILP/MILP: At each node of the branch-decision tree:
  - solve the LP relaxation of the (sub-)problem associated with the the current decision node;
  - if the solution is not integer, separate a fractional variable to get 2 new sub-problems (2 new decision nodes);
  - continue until all decision nodes have been explored.
- **Branch-and-Cut:** ...(Branch-and-Bound) but
  - if the solution is not integer, before separating, add cutting planes to strengthen the relaxation, possibly finding an integer solution or improving on the lower bound value.
  - if the solution remains not integer, separate.
- **Branch-and-Price:** ...(Branch-and-Bound) but
  - use *column generation* to solve the LP relaxation at each node:
  - initially, only a subset of columns is included in the LP relaxation (*Restricted Master Problem*);
  - an auxiliary problem (*Pricing Problem*) is used to check optimality and to find columns to be added to improve the LP solution value;
- **Branch-and-Price-and-Cut** = Branch-and-Bound + column generation + cutting planes.
- For the BPP and the CSP, all Branch-and-Price (-and-Cut) algorithms are based on the *set covering formulation* and the solution of its continuous relaxation through column generation (seminal work by Gilmore and Gomory).

# Set covering formulation

## Set covering formulation

- Enumeration of the set $P$ of all *patterns* $p$ (combinations of items that can fit into a bin).

## Set covering formulation

- Enumeration of the set $P$ of all *patterns* $p$ (combinations of items that can fit into a bin).
- **For the CSP:** pattern $p \equiv$ integer array $(a_{1p}, a_{2p}, \dots)$, with $a_{jp} =$ number of copies of item $j$ contained in pattern $p$, satisfying $\sum_{j=1}^{m} a_{jp} w_j \le c$ and $a_{jp} \ge 0$, integer $\forall j$.

# Set covering formulation

- Enumeration of the set $P$ of all *patterns* $p$ (combinations of items that can fit into a bin).
- **For the CSP:** pattern $p \equiv$ integer array $(a_{1p}, a_{2p}, \dots)$, with $a_{jp} =$ number of copies of item $j$ contained in pattern $p$, satisfying $\displaystyle\sum_{j=1}^{m} a_{jp} w_j \le c$ and $a_{jp} \ge 0$, integer $\forall j$.
- Let $y_p =$ number of times pattern $p$ is used. **Set covering** formulation of the CSP:

# Set covering formulation

- Enumeration of the set $P$ of all *patterns* $p$ (combinations of items that can fit into a bin).
- **For the CSP:** pattern $p \equiv$ integer array $(a_{1p}, a_{2p}, \ldots)$, with $a_{jp} =$ number of copies of item $j$ contained in pattern $p$, satisfying $\sum_{j=1}^{m} a_{jp} w_j \leq c$ and $a_{jp} \geq 0$, integer $\forall j$.
- Let $y_p =$ number of times pattern $p$ is used. **Set covering** formulation of the CSP:

$$\min \quad \sum_{p \in P} y_p \tag{11}$$

$$\text{s.t.} \quad \sum_{p \in P} a_{jp} y_p \geq d_j \quad (j = 1, \ldots, m), \tag{12}$$

$$y_p \geq 0 \text{ and integer } (p \in P). \tag{13}$$

# Set covering formulation

- Enumeration of the set $P$ of all *patterns* $p$ (combinations of items that can fit into a bin).
- **For the CSP:** pattern $p \equiv$ integer array $(a_{1p}, a_{2p}, \dots)$, with $a_{jp}$ = number of copies of item $j$ contained in pattern $p$, satisfying $\sum_{j=1}^{m} a_{jp} w_j \leq c$ and $a_{jp} \geq 0$, integer $\forall j$.
- Let $y_p$ = number of times pattern $p$ is used. **Set covering** formulation of the CSP:

$$\min \sum_{p \in P} y_p \tag{11}$$

$$\text{s.t.} \quad \sum_{p \in P} a_{jp} y_p \geq d_j \qquad (j = 1, \dots, m), \tag{12}$$

$$y_p \geq 0 \text{ and integer } (p \in P). \tag{13}$$

- Similarly **for the BPP**: (i) $p \equiv$ binary array, $y_p$ binary ($= 1$ iff pattern $p$ is used for a bin):

# Set covering formulation

- Enumeration of the set $P$ of all *patterns* $p$ (combinations of items that can fit into a bin).
- **For the CSP:** pattern $p \equiv$ integer array $(a_{1p}, a_{2p}, \dots)$, with $a_{jp}$ = number of copies of item $j$ contained in pattern $p$, satisfying $\sum_{j=1}^{m} a_{jp} w_j \le c$ and $a_{jp} \ge 0$, integer $\forall j$.
- Let $y_p$ = number of times pattern $p$ is used. **Set covering** formulation of the CSP:

$$\min \quad \sum_{p \in P} y_p \tag{11}$$

$$\text{s.t.} \quad \sum_{p \in P} a_{jp} y_p \ge d_j \quad (j = 1, \dots, m), \tag{12}$$

$$y_p \ge 0 \text{ and integer } (p \in P). \tag{13}$$

- Similarly **for the BPP**: (i) $p \equiv$ binary array, $y_p$ binary ($= 1$ iff pattern $p$ is used for a bin):

$$\sum_{p \in P} a_{jp} y_p \ge 1 \; (j = 1, \dots, n) \tag{14}$$

# Set covering formulation

- Enumeration of the set $P$ of all *patterns* $p$ (combinations of items that can fit into a bin).
- **For the CSP:** pattern $p \equiv$ integer array $(a_{1p}, a_{2p}, \dots)$, with $a_{jp} =$ number of copies of item $j$ contained in pattern $p$, satisfying $\sum_{j=1}^{m} a_{jp} w_j \le c$ and $a_{jp} \ge 0$, integer $\forall j$.
- Let $y_p =$ number of times pattern $p$ is used. **Set covering** formulation of the CSP:

$$\min \quad \sum_{p \in P} y_p \tag{11}$$

$$\text{s.t.} \quad \sum_{p \in P} a_{jp} y_p \ge d_j \quad (j = 1, \dots, m), \tag{12}$$

$$y_p \ge 0 \text{ and integer } (p \in P). \tag{13}$$

- Similarly **for the BPP**: (i) $p \equiv$ binary array, $y_p$ binary ($= 1$ iff pattern $p$ is used for a bin):

$$\sum_{p \in P} a_{jp} y_p \ge 1 \; (j = 1, \dots, n) \tag{14}$$

- **the number of feasible patterns is exponential**
$\Longrightarrow$ **the number of columns of the LP relaxation is exponential**

## Set covering formulation

- Enumeration of the set $P$ of all *patterns* $p$ (combinations of items that can fit into a bin).
- **For the CSP:** pattern $p \equiv$ integer array $(a_{1p}, a_{2p}, \dots)$, with $a_{jp}$ = number of copies of item $j$ contained in pattern $p$, satisfying $\sum_{j=1}^{m} a_{jp} w_j \le c$ and $a_{jp} \ge 0$, integer $\forall j$.
- Let $y_p$ = number of times pattern $p$ is used. **Set covering** formulation of the CSP:

$$\min \sum_{p \in P} y_p \tag{11}$$

$$\text{s.t.} \quad \sum_{p \in P} a_{jp} y_p \ge d_j \quad (j = 1, \dots, m), \tag{12}$$

$$y_p \ge 0 \text{ and integer } (p \in P). \tag{13}$$

- Similarly **for the BPP**: (i) $p \equiv$ binary array, $y_p$ binary ($= 1$ iff pattern $p$ is used for a bin):

$$\sum_{p \in P} a_{jp} y_p \ge 1 \; (j = 1, \dots, n) \tag{14}$$

- **the number of feasible patterns is exponential**
  $\Longrightarrow$ **the number of columns of the LP relaxation is exponential** $\Longrightarrow$ **Column generation**

# Column generation

# Column generation

- Heuristically initialize the LP relaxation with a subset of patterns $P' \subset P$ (**Restricted Master Problem, RMP**):

## Column generation

- Heuristically initialize the LP relaxation with a subset of patterns $P' \subset P$ (**Restricted Master Problem, RMP**):

$$\min \sum_{p \in P'} y_p \tag{15}$$

$$\text{s.t.} \quad \sum_{p \in P'} a_{jp} y_p \geq d_j \quad (j = 1, \ldots, m), \tag{16}$$

$$y_p \geq 0 \qquad (p \in P').$$

# Column generation

- Heuristically initialize the LP relaxation with a subset of patterns $P' \subset P$ (**Restricted Master Problem, RMP**):

$$\min \quad \sum_{p \in P'} y_p \tag{15}$$

$$\text{s.t.} \quad \sum_{p \in P'} a_{jp} y_p \geq d_j \quad (j = 1, \dots, m), \tag{16}$$

$$y_p \geq 0 \qquad (p \in P'). \tag{17}$$

- Solve (15)-(17) and let $\pi_j$ be the dual variables associated with the $j$th constraint (16).
- **Pricing:** find a column $p \notin P'$ that could reduce the objective function value:

# Column generation

- Heuristically initialize the LP relaxation with a subset of patterns $P' \subset P$ (**Restricted Master Problem, RMP**):

$$\min \sum_{p \in P'} y_p \tag{15}$$

$$\text{s.t.} \quad \sum_{p \in P'} a_{jp} y_p \geq d_j \quad (j = 1, \dots, m), \tag{16}$$

$$y_p \geq 0 \quad (p \in P'). \tag{17}$$

- Solve (15)-(17) and let $\pi_j$ be the dual variables associated with the $j$th constraint (16).
- **Pricing:** find a column $p \notin P'$ that could reduce the objective function value:
  - find the column with the most negative reduced cost (**Slave Problem** (SP)) by solving an associated *knapsack problem* in the dual variables.
  - if the SP finds such a column (pattern), then add the corresponding column to the RMP.

## Column generation

- Heuristically initialize the LP relaxation with a subset of patterns $P' \subset P$ (**Restricted Master Problem, RMP**):

$$\min \sum_{p \in P'} y_p \qquad (15)$$

$$\text{s.t.} \quad \sum_{p \in P'} a_{jp} y_p \geq d_j \quad (j = 1, \ldots, m), \qquad (16)$$

$$y_p \geq 0 \quad (p \in P'). \qquad (17)$$

- Solve (15)-(17) and let $\pi_j$ be the dual variables associated with the $j$th constraint (16).
- **Pricing:** find a column $p \notin P'$ that could reduce the objective function value:
  - find the column with the most negative reduced cost (**Slave Problem** (SP)) by solving an associated *knapsack problem* in the dual variables.
  - if the SP finds such a column (pattern), then add the corresponding column to the RMP.
- Iterate until no column with negative reduced cost is found (optimal solution).

## Column generation

- Heuristically initialize the LP relaxation with a subset of patterns $P' \subset P$ (**Restricted Master Problem, RMP**):

$$
\begin{aligned}
\min \quad & \sum_{p \in P'} y_p && && (15) \\
\text{s.t.} \quad & \sum_{p \in P'} a_{jp} y_p \geq d_j && (j = 1, \dots, m), && (16) \\
& y_p \geq 0 && (p \in P'). && (17)
\end{aligned}
$$

- Solve (15)-(17) and let $\pi_j$ be the dual variables associated with the $j$th constraint (16).
- **Pricing:** find a column $p \notin P'$ that could reduce the objective function value:
  - find the column with the most negative reduced cost (**Slave Problem** (SP)) by solving an associated *knapsack problem* in the dual variables.
  - if the SP finds such a column (pattern), then add the corresponding column to the RMP.
- Iterate until no column with negative reduced cost is found (optimal solution).
- Huge number of **Branch-and-Price(-and-Cut) algorithms** in the Nineties and the Noughties;

# Column generation

- Heuristically initialize the LP relaxation with a subset of patterns $P' \subset P$ (**Restricted Master Problem, RMP**):

$$\min \sum_{p \in P'} y_p \tag{15}$$

$$\text{s.t.} \sum_{p \in P'} a_{jp} y_p \geq d_j \quad (j = 1, \ldots, m), \tag{16}$$

$$y_p \geq 0 \qquad (p \in P'). \tag{17}$$

- Solve (15)-(17) and let $\pi_j$ be the dual variables associated with the $j$th constraint (16).
- **Pricing:** find a column $p \notin P'$ that could reduce the objective function value:
  - find the column with the most negative reduced cost (**Slave Problem** (SP)) by solving an associated *knapsack problem* in the dual variables.
  - if the SP finds such a column (pattern), then add the corresponding column to the RMP.
- Iterate until no column with negative reduced cost is found (optimal solution).
- Huge number of **Branch-and-Price(-and-Cut) algorithms** in the Nineties and the Noughties;
- Most efficient algorithm (and C++ computer code): Belov and Scheithauer (2006).

## Parenthesis: IRUP and MIRUP (BPP and CSP)

## Parenthesis: IRUP and MIRUP (BPP and CSP)

- $L_{\mathsf{LP}} =$ solution value of the LP relaxation of the set covering formulation;

# Parenthesis: IRUP and MIRUP (BPP and CSP)

- $L_{\mathsf{LP}}$ = solution value of the LP relaxation of the set covering formulation;
- $z_{\mathsf{opt}}$ = optimal solution value;

## Parenthesis: IRUP and MIRUP (BPP and CSP)

- $L_{\text{LP}} =$ solution value of the LP relaxation of the set covering formulation;
- $z_{\text{opt}} =$ optimal solution value;
- **IRUP** (*Integer Round-Up Property*) **conjecture:** $z_{\text{opt}} = \lceil L_{\text{LP}} \rceil$.

# Parenthesis: IRUP and MIRUP (BPP and CSP)

- $L_{\text{LP}} =$ solution value of the LP relaxation of the set covering formulation;
- $z_{\text{opt}} =$ optimal solution value;
- **IRUP** (*Integer Round-Up Property*) **conjecture:** $z_{\text{opt}} = \lceil L_{\text{LP}} \rceil$.
- **Disproved** by Marcotte (1986) (instance with $n = 24$ and $c = 3,397,386,255$).

## Parenthesis: IRUP and MIRUP (BPP and CSP)

- $L_{\text{LP}}$ = solution value of the LP relaxation of the set covering formulation;
- $z_{\text{opt}}$ = optimal solution value;
- **IRUP** (*Integer Round-Up Property*) **conjecture:** $z_{\text{opt}} = \lceil L_{\text{LP}} \rceil$.
- **Disproved** by Marcotte (1986) (instance with $n = 24$ and $c = 3, 397, 386, 255$).
- **MIRUP** (*Modified IRUP*) **conjecture:** $z_{\text{opt}} \leq \lceil L_{\text{LP}} \rceil + 1$.

# Parenthesis: IRUP and MIRUP (BPP and CSP)

- $L_{\text{LP}}$ = solution value of the LP relaxation of the set covering formulation;
- $z_{\text{opt}}$ = optimal solution value;
- **IRUP** (*Integer Round-Up Property*) **conjecture:** $z_{\text{opt}} = \lceil L_{\text{LP}} \rceil$.
- **Disproved** by Marcotte (1986) (instance with $n = 24$ and $c = 3,397,386,255$).
- **MIRUP** (*Modified IRUP*) **conjecture:** $z_{\text{opt}} \leq \lceil L_{\text{LP}} \rceil + 1$.
- **Conjecture open**.

# Pseudo-Polynomial Formulations

## Pseudo-Polynomial Formulations

- The number of variables and constraints depends on the number of items **and** on the **bin capacity**.

# Pseudo-Polynomial Formulations

- The number of variables and constraints depends on the number of items **and** on the **bin capacity**.

## One-cut formulation

- Independently developed by Rao in 1976 and by Dyckhoff in 1981.

# Pseudo-Polynomial Formulations

- The number of variables and constraints depends on the number of items **and** on the **bin capacity**.

## One-cut formulation

- Independently developed by Rao in 1976 and by Dyckhoff in 1981.
- Basic idea (for the **CSP**): simulate the physical cutting process:
  - divide an ideal bin into **two pieces**, where

# Pseudo-Polynomial Formulations

- The number of variables and constraints depends on the number of items **and** on the **bin capacity**.

## One-cut formulation

- Independently developed by Rao in 1976 and by Dyckhoff in 1981.
- Basic idea (for the **CSP**): simulate the physical cutting process:
  - divide an ideal bin into **two pieces**, where
  - the **left piece** is an item that has been cut;
  - the **right piece** is either another item

    or a residual that can be re-used to produce other items.

# Pseudo-Polynomial Formulations

- The number of variables and constraints depends on the number of items **and** on the **bin capacity**.

## One-cut formulation

- Independently developed by Rao in 1976 and by Dyckhoff in 1981.
- Basic idea (for the **CSP**): simulate the physical cutting process:
  - divide an ideal bin into **two pieces**, where
  - the **left piece** is an item that has been cut;
  - the **right piece** is either another item

    or a residual that can be re-used to produce other items.
  - Iterate the process on cutting residuals or new bins, until all demands are fulfilled.

# Pseudo-Polynomial Formulations

- The number of variables and constraints depends on the number of items **and** on the **bin capacity**.

## One-cut formulation

- Independently developed by Rao in 1976 and by Dyckhoff in 1981.
- Basic idea (for the **CSP**): simulate the physical cutting process:
  - divide an ideal bin into **two pieces**, where
  - the **left piece** is an item that has been cut;
  - the **right piece** is either another item

    or a residual that can be re-used to produce other items.
  - Iterate the process on cutting residuals or new bins, until all demands are fulfilled.
  - **Integer variables** $x_{pq}$ **=** number of times a bin, or a residual of width $p$, is cut into a left piece of width $q$ and a right piece of width $p - q$.

# Pseudo-Polynomial Formulations

- The number of variables and constraints depends on the number of items **and** on the **bin capacity**.

## One-cut formulation

- Independently developed by Rao in 1976 and by Dyckhoff in 1981.
- Basic idea (for the **CSP**): simulate the physical cutting process:
  - divide an ideal bin into **two pieces**, where
  - the **left piece** is an item that has been cut;
  - the **right piece** is either another item

    or a residual that can be re-used to produce other items.
  - Iterate the process on cutting residuals or new bins, until all demands are fulfilled.
  - **Integer variables** $x_{pq}$ **=** number of times a bin, or a residual of width $p$, is cut into a left piece of width $q$ and a right piece of width $p - q$.
- The resulting ILP model has $O(mc)$ variables and $O(c)$ constraints.

# DP-flow formulation

# DP-flow formulation

- Cambazard and O'Sullivan (2010): Basic idea (for the **BPP**):
  - associate variables with the decisions taken in a classical *dynamic programming (DP) table*;

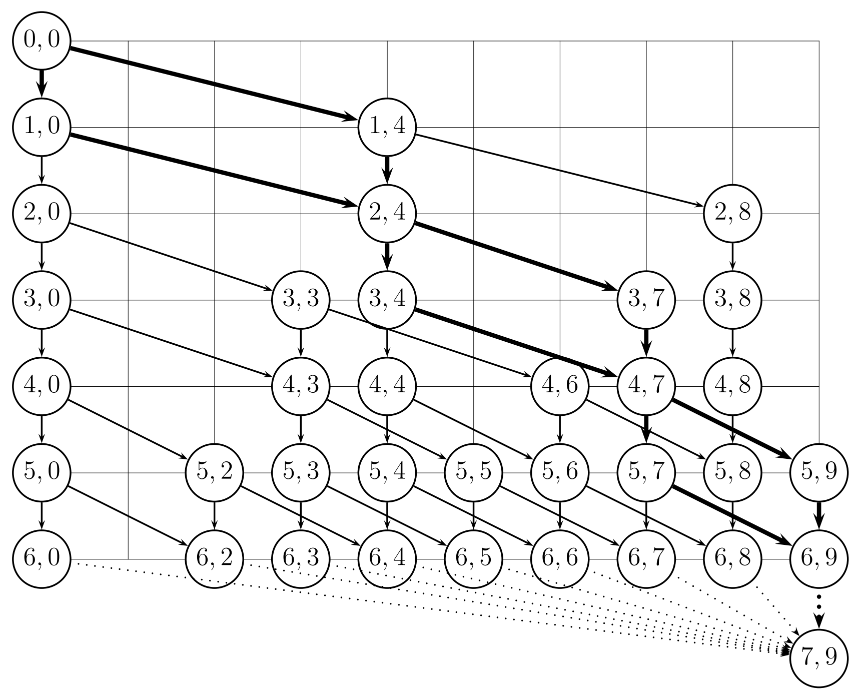
# DP-flow formulation

- Cambazard and O'Sullivan (2010): Basic idea (for the **BPP**):
  - associate variables with the decisions taken in a classical *dynamic programming (DP) table*;
  - DP states $\leftrightarrow$ graph: **path** from initial to terminal node $=$ feasible filling of a bin.

# DP-flow formulation

- Cambazard and O'Sullivan (2010): Basic idea (for the **BPP**):
  - associate variables with the decisions taken in a classical *dynamic programming (DP) table*;
  - DP states $\leftrightarrow$ graph: **path** from initial to terminal node = feasible filling of a bin.
- Example: $n = 6$, $c = 9$, $w = (4, 4, 3, 3, 2, 2)$:
  $[j, d]$ $(j = 0, .., n; d = 0, .., c)$: [decisions taken up to item $j$, partial bin filling $d$ units].

**Figure 1** DP-flow graph construction for Example 1

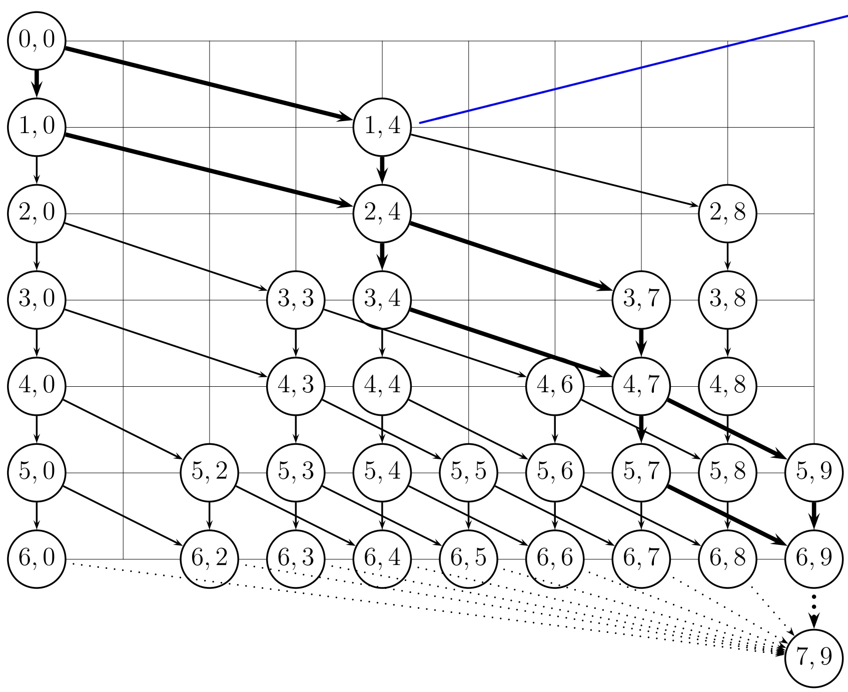


# DP-flow formulation

- Cambazard and O'Sullivan (2010): Basic idea (for the **BPP**):
  - associate variables with the decisions taken in a classical *dynamic programming (DP) table*;
  - DP states $\leftrightarrow$ graph: **path** from initial to terminal node = feasible filling of a bin.
- Example: $n = 6$, $c = 9$, $w = (4, 4, 3, 3, 2, 2)$:
  $[j, d]$ $(j = 0, .., n; d = 0, .., c)$: [decisions taken up to item $j$, partial bin filling $d$ units].

**Figure 1** DP-flow graph construction for Example 1

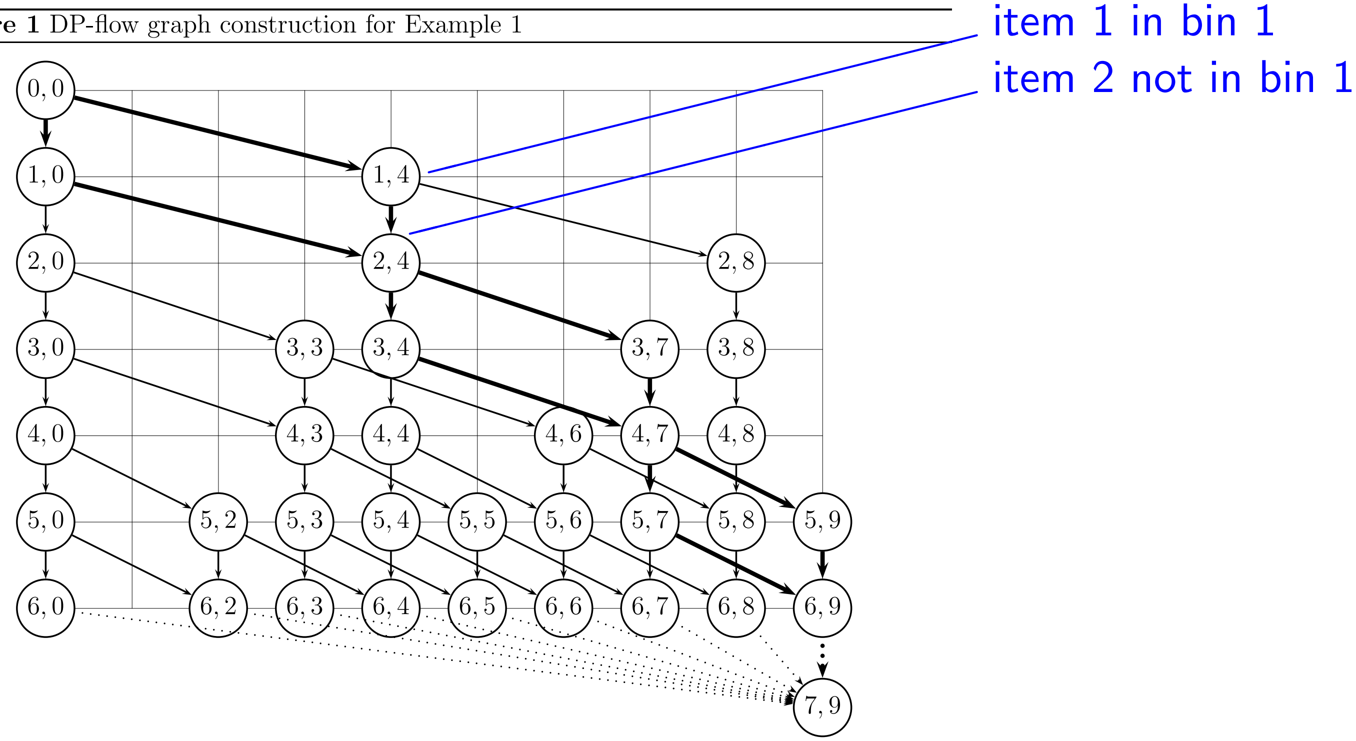
item 1 in bin 1



# DP-flow formulation

- Cambazard and O'Sullivan (2010): Basic idea (for the **BPP**):
  - associate variables with the decisions taken in a classical *dynamic programming (DP) table*;
  - DP states $\leftrightarrow$ graph: **path** from initial to terminal node = feasible filling of a bin.
- Example: $n = 6$, $c = 9$, $w = (4, 4, 3, 3, 2, 2)$:
  $[j, d]$ $(j = 0, .., n; d = 0, .., c)$: [decisions taken up to item $j$, partial bin filling $d$ units].

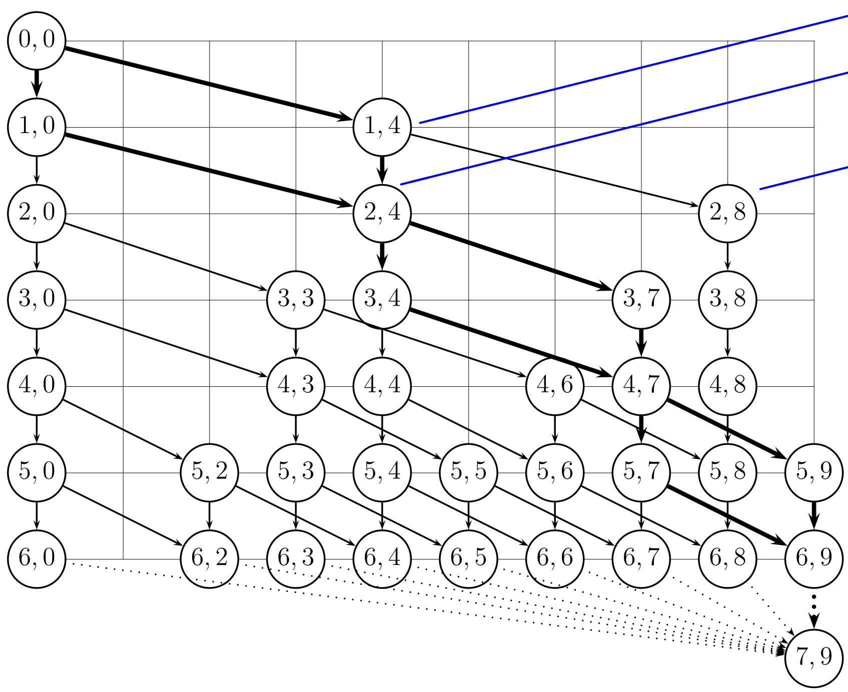
**Figure 1** DP-flow graph construction for Example 1



# DP-flow formulation

- Cambazard and O'Sullivan (2010): Basic idea (for the **BPP**):
  - associate variables with the decisions taken in a classical *dynamic programming (DP) table*;
  - DP states $\leftrightarrow$ graph: **path** from initial to terminal node = feasible filling of a bin.
- Example: $n = 6$, $c = 9$, $w = (4, 4, 3, 3, 2, 2)$:
  $[j, d]$ $(j = 0, .., n; d = 0, .., c)$: [decisions taken up to item $j$, partial bin filling $d$ units].

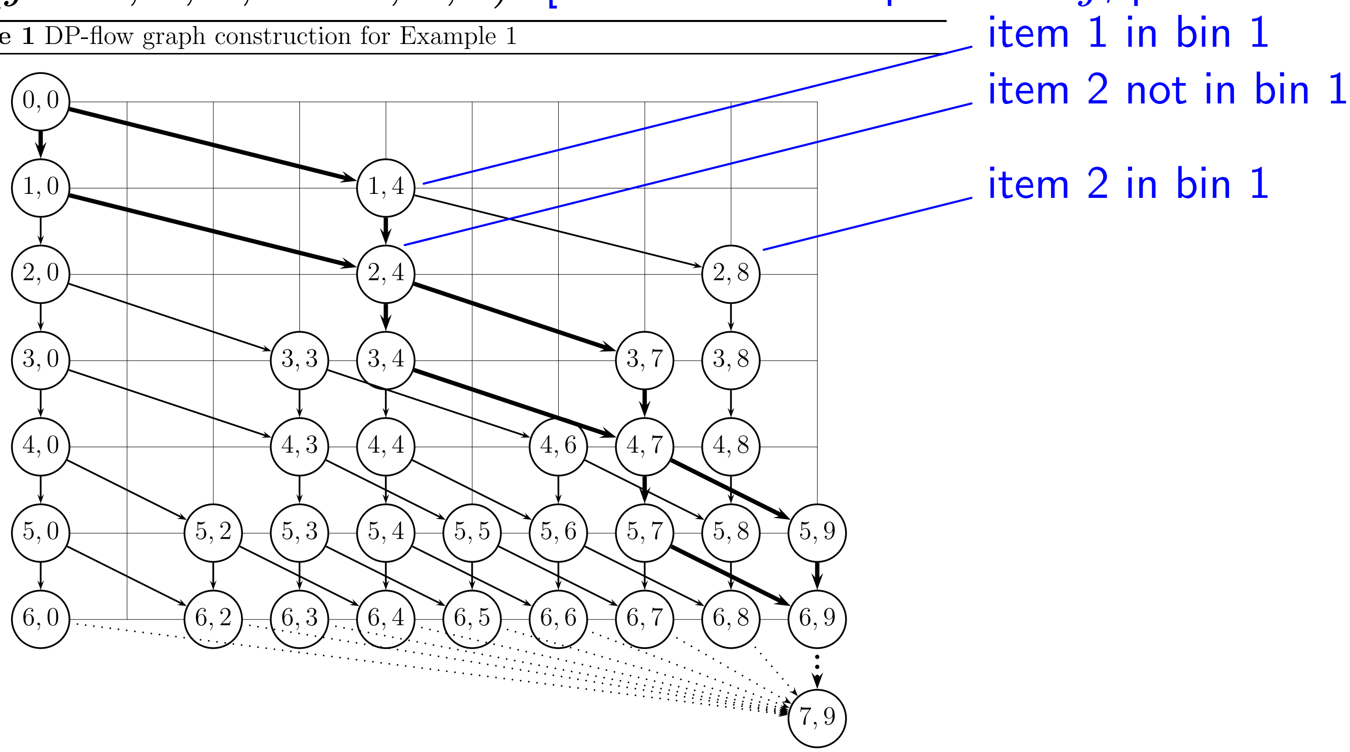
**Figure 1** DP-flow graph construction for Example 1



## DP-flow formulation

- Cambazard and O'Sullivan (2010): Basic idea (for the **BPP**):
  - associate variables with the decisions taken in a classical *dynamic programming (DP) table*;
  - DP states $\leftrightarrow$ graph: **path** from initial to terminal node = feasible filling of a bin.
- Example: $n = 6$, $c = 9$, $w = (4, 4, 3, 3, 2, 2)$:
  $[j, d]$ $(j = 0, .., n; d = 0, .., c)$: [decisions taken up to item $j$, partial bin filling $d$ units].

**Figure 1** DP-flow graph construction for Example 1



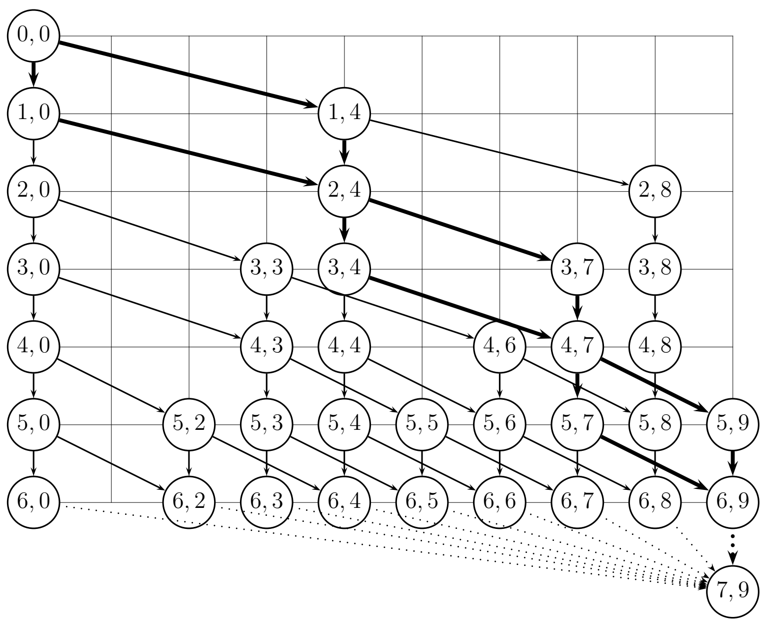

- Network Flow-type model to minimize the number of paths. $O(nc)$ variables and constraints.

# Arc-flow formulation

# Arc-flow formulation

- Valério de Carvalho (1999) anticipated DP-flow (but Wolsey (1977) anticipated everybody): vertically shrunk the DP graph: states with the same partial bin filling $\longrightarrow$ single state:

**Figure 1** DP-flow graph construction for Example 1



# Arc-flow formulation

- Valério de Carvalho (1999) anticipated DP-flow (but Wolsey (1977) anticipated everybody): vertically shrunk the DP graph: states with the same partial bin filling $\longrightarrow$ single state:

**Figure 1** DP-flow graph construction for Example 1

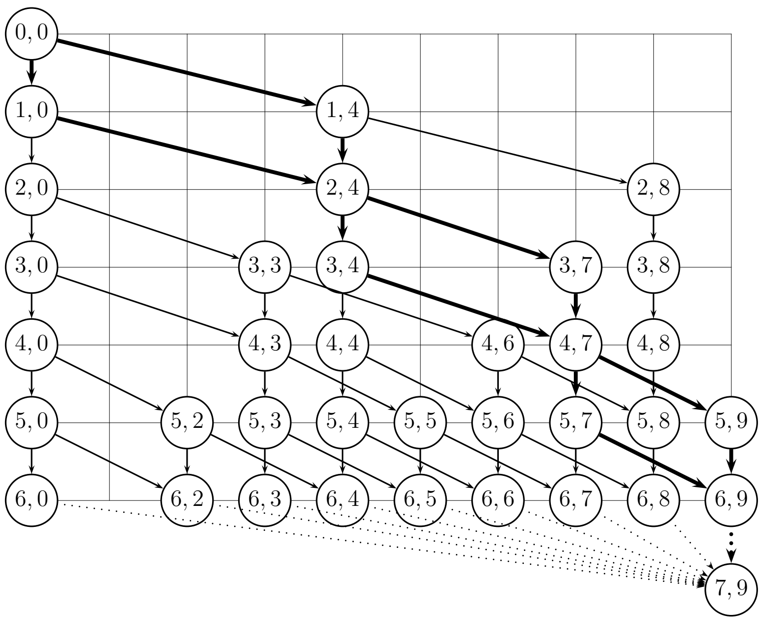
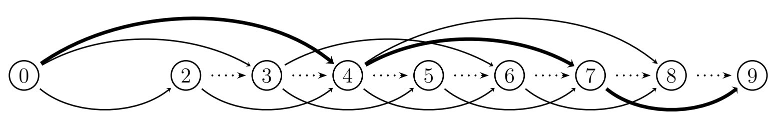



**Figure 2** Arc-flow representation of the graph of Figure 1



# Arc-flow formulation

- Valério de Carvalho (1999) anticipated DP-flow (but Wolsey (1977) anticipated everybody): vertically shrunk the DP graph: states with the same partial bin filling $\longrightarrow$ single state:

**Figure 1** DP-flow graph construction for Example 1

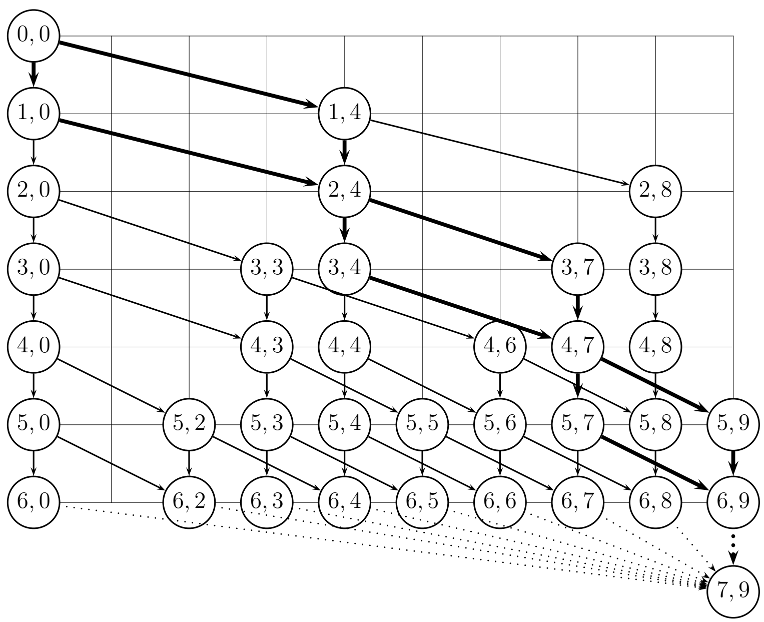
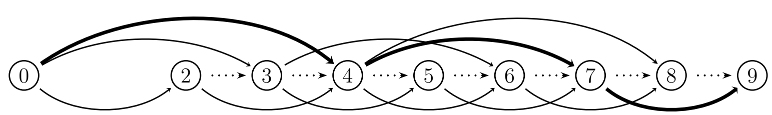



**Figure 2** Arc-flow representation of the graph of Figure 1




- **CSP** modeled as a network flow problem;

# Arc-flow formulation

- Valério de Carvalho (1999) anticipated DP-flow (but Wolsey (1977) anticipated everybody): vertically shrunk the DP graph: states with the same partial bin filling $\rightarrow$ single state:

**Figure 1** DP-flow graph construction for Example 1

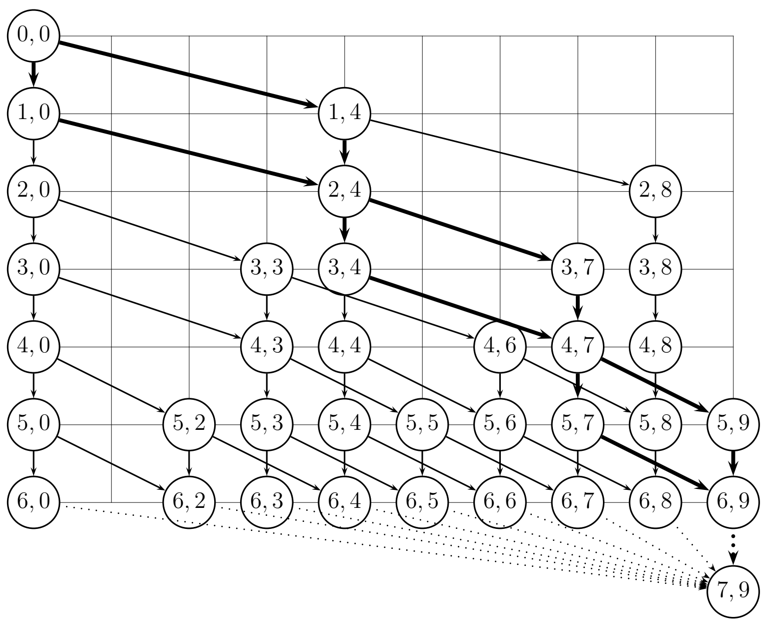
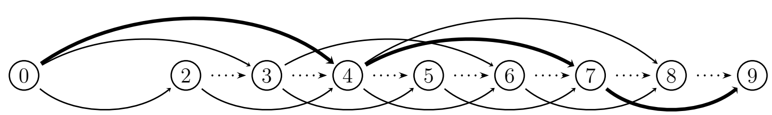


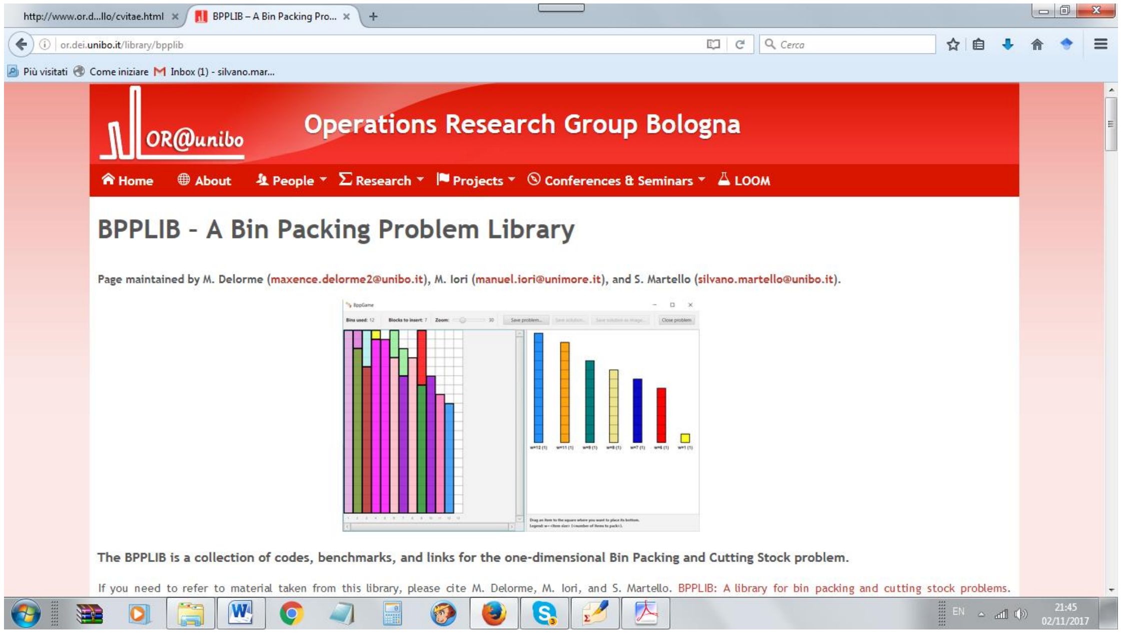

**Figure 2** Arc-flow representation of the graph of Figure 1




- **CSP** modeled as a network flow problem;
- Brandão and Pedroso (2016): alternative arc-flow formulation, very effective code VPSOLVER.

# Computer codes and the BPPLIB

# Computer codes and the BPPLIB



# Computer codes and the BPPLIB


You will find (from this talk):
MTP (branch-and-bound, Fortran)
**BISON** (branch-and-bound, MTP + Tabu Search, Pascal)
**BELOV** (branch-and-cut-and-price, C++ & Cplex)
ONECUT (pseudo-polynomial, C++ & Cplex/SCIP))
**ARCFLOW** (pseudo-polynomial, C++ & Cplex/SCIP)
DPFLOW (pseudo-polynomial, C++ & Cplex/SCIP)
**VPSOLVER** (pseudo-polynomial, C++ & Gurobi)

# Computer codes and the BPPLIB


You will find (from this talk):
MTP (branch-and-bound, Fortran)
**BISON** (branch-and-bound, MTP + Tabu Search, Pascal)
**BELOV** (branch-and-cut-and-price, C++ & Cplex)
ONECUT (pseudo-polynomial, C++ & Cplex/SCIP))
**ARCFLOW** (pseudo-polynomial, C++ & Cplex/SCIP)
DPFLOW (pseudo-polynomial, C++ & Cplex/SCIP)
**VPSOLVER** (pseudo-polynomial, C++ & Gurobi)

Other codes, benchmarks, links, BibTeX file, interactive visual solver.

# Experimental evaluation (BPP)

# Experimental evaluation (BPP)

Number of **literature** instances solved in less than **10 minutes**

| Set | # inst. | BISON | BELOV | ARCFLOW | VPSOLVER |
|---|---|---|---|---|---|
| Falkenauer U | 74 | 50 | **74** | **74** | **74** |
| Falkenauer T | 80 | 47 | **80** | **80** | **80** |
| Scholl 1 | 323 | 290 | **323** | **323** | **323** |
| Scholl 2 | 244 | 234 | **244** | 231 | 242 |
| Scholl 3 | 10 | 3 | **10** | 0 | **10** |
| Wäscher | 17 | 10 | **17** | 4 | 13 |
| Schwerin 1 | 100 | **100** | **100** | **100** | **100** |
| Schwerin 2 | 100 | 63 | **100** | **100** | **100** |
| Hard28 | 28 | 0 | **28** | 26 | 26 |
| Total | 976 | 797 | **976** | 938 | 968 |

# Experimental evaluation (BPP)

## Number of **literature** instances solved in less than **10 minutes**

| Set | # inst. | BISON | BELOV | ARCFLOW | VPSOLVER |
|---|---|---|---|---|---|
| Falkenauer U | 74 | 50 | **74** | **74** | **74** |
| Falkenauer T | 80 | 47 | **80** | **80** | **80** |
| Scholl 1 | 323 | 290 | **323** | **323** | **323** |
| Scholl 2 | 244 | 234 | **244** | 231 | 242 |
| Scholl 3 | 10 | 3 | **10** | 0 | **10** |
| Wäscher | 17 | 10 | **17** | 4 | 13 |
| Schwerin 1 | 100 | **100** | **100** | **100** | **100** |
| Schwerin 2 | 100 | 63 | **100** | **100** | **100** |
| Hard28 | 28 | 0 | **28** | 26 | 26 |
| Total | 976 | 797 | **976** | 938 | 968 |

## Number of **random** instances solved in less than **10 minutes**

| $n$ | # inst. | BISON | BELOV | ARCFLOW | VPSOLVER |
|---|---|---|---|---|---|
| 50 | 165 | **165** | **165** | **165** | **165** |
| 100 | 271 | 261 | **271** | **271** | **271** |
| 200 | 359 | 299 | **359** | **359** | **359** |
| 300 | 393 | 269 | **393** | **393** | **393** |
| 400 | 425 | 250 | **425** | **425** | **425** |
| 500 | 414 | 212 | **414** | **414** | **414** |
| 750 | 433 | 217 | **433** | 431 | **433** |
| 1000 | 441 | 200 | **441** | 434 | **441** |
| Total | 2901 | 1873 | **2901** | 2892 | **2901** |

# Difficult instances

# Difficult instances

Number of **difficult (ANI)** instances, out of **50**, solved in less than **1 hour (average absolute gap)**

| $n$ | $\overline{c}$ | BISON | BELOV | ARCFLOW | VPSOLVER |
|---|---|---|---|---|---|
| 201 | 2500 | 0 (1.0) | **50** (0.0) | 16 (0.7) | 47 (0.1) |
| 402 | 10000 | 0 (1.0) | 1 (1.0) | 0 (1.0) | 6 (0.9) |
| 600 | 20000 | - | (1.0) | - | 0 (1.0) |
| 801 | 40000 | - | (1.0) | - | 0 (1.0) |
| 1002 | 80000 | - | - | - | - |
| Overall | | 0 (1.0) | 51 (0.7) | 16 (0.8) | 53 (0.7) |

# Difficult instances

Number of **difficult (ANI)** instances, out of **50**, solved in less than **1 hour (average absolute gap)**

| $n$ | $\overline{c}$ | BISON | BELOV | ARCFLOW | VPSOLVER |
|---|---|---|---|---|---|
| 201 | 2500 | 0 (1.0) | **50** (0.0) | 16 (0.7) | 47 (0.1) |
| 402 | 10000 | 0 (1.0) | 1 (1.0) | 0 (1.0) | 6 (0.9) |
| 600 | 20000 | - | (1.0) | - | 0 (1.0) |
| 801 | 40000 | - | (1.0) | - | 0 (1.0) |
| 1002 | 80000 | - | - | - | - |
| Overall | | 0 (1.0) | 51 (0.7) | 16 (0.8) | 53 (0.7) |

**A final comment:**

- Originally (ARCFLOW, 1999) pseudo-polynomial formulations were seen as theoretical results and rarely directly used in practice as ILP formulations (too many variables and constraints).

# Difficult instances

Number of **difficult (ANI)** instances, out of **50**, solved in less than **1 hour (average absolute gap)**

| $n$ | $\overline{c}$ | BISON | BELOV | ARCFLOW | VPSOLVER |
|---|---|---|---|---|---|
| 201 | 2500 | 0 (1.0) | **50** (0.0) | 16 (0.7) | 47 (0.1) |
| 402 | 10000 | 0 (1.0) | 1 (1.0) | 0 (1.0) | 6 (0.9) |
| 600 | 20000 | - | (1.0) | - | 0 (1.0) |
| 801 | 40000 | - | (1.0) | - | 0 (1.0) |
| 1002 | 80000 | - | - | - | - |
| Overall | | 0 (1.0) | 51 (0.7) | 16 (0.8) | 53 (0.7) |

**A final comment:**

- Originally (ARCFLOW, 1999) pseudo-polynomial formulations were seen as theoretical results and rarely directly used in practice as ILP formulations (too many variables and constraints).
- Nowadays they are extremely competitive in practice. **Why?**

# Difficult instances

Number of **difficult (ANI)** instances, out of **50**, solved in less than **1 hour (average absolute gap)**

| $n$ | $\overline{c}$ | BISON | BELOV | ARCFLOW | VPSOLVER |
|---|---|---|---|---|---|
| 201 | 2500 | 0 (1.0) | **50** (0.0) | 16 (0.7) | 47 (0.1) |
| 402 | 10000 | 0 (1.0) | 1 (1.0) | 0 (1.0) | 6 (0.9) |
| 600 | 20000 | - | (1.0) | - | 0 (1.0) |
| 801 | 40000 | - | (1.0) | - | 0 (1.0) |
| 1002 | 80000 | - | - | - | - |
| Overall | | 0 (1.0) | 51 (0.7) | 16 (0.8) | 53 (0.7) |

## A final comment:

- Originally (ARCFLOW, 1999) pseudo-polynomial formulations were seen as theoretical results and rarely directly used in practice as ILP formulations (too many variables and constraints).
- Nowadays they are extremely competitive in practice. **Why?**
- 20 selected random instances ($n \in [300, 1000]$, $c \in [400, 1000]$;
  ARCFLOW: $\#$ constraints $\in [482, 1093]$, #variables $\in [32\,059, 111\,537]$);

# Difficult instances

Number of **difficult (ANI)** instances, out of **50**, solved in less than **1 hour (average absolute gap)**

| $n$ | $\overline{c}$ | BISON | BELOV | ARCFLOW | VPSOLVER |
|---|---|---|---|---|---|
| 201 | 2500 | 0 (1.0) | **50** (0.0) | 16 (0.7) | 47 (0.1) |
| 402 | 10000 | 0 (1.0) | 1 (1.0) | 0 (1.0) | 6 (0.9) |
| 600 | 20000 | - | (1.0) | - | 0 (1.0) |
| 801 | 40000 | - | (1.0) | - | 0 (1.0) |
| 1002 | 80000 | - | - | - | - |
| Overall | | 0 (1.0) | 51 (0.7) | 16 (0.8) | 53 (0.7) |

**A final comment:**

- Originally (ARCFLOW, 1999) pseudo-polynomial formulations were seen as theoretical results and rarely directly used in practice as ILP formulations (too many variables and constraints).
- Nowadays they are extremely competitive in practice. **Why?**
- 20 selected random instances ($n \in [300, 1000]$, $c \in [400, 1000]$;
  ARCFLOW: $\#$ constraints $\in [482, 1093]$, #variables $\in [32\,059, 111\,537]$);
  **8 versions of CPLEX**: number of solved instances [average CPU time]:

## Difficult instances

Number of **difficult (ANI)** instances, out of **50**, solved in less than **1 hour (average absolute gap)**

| $n$ | $\overline{c}$ | BISON | BELOV | ARCFLOW | VPSOLVER |
|---|---|---|---|---|---|
| 201 | 2500 | 0 (1.0) | **50** (0.0) | 16 (0.7) | 47 (0.1) |
| 402 | 10000 | 0 (1.0) | 1 (1.0) | 0 (1.0) | 6 (0.9) |
| 600 | 20000 | - | (1.0) | - | 0 (1.0) |
| 801 | 40000 | - | (1.0) | - | 0 (1.0) |
| 1002 | 80000 | - | - | - | - |
| Overall | | 0 (1.0) | 51 (0.7) | 16 (0.8) | 53 (0.7) |

### A final comment:

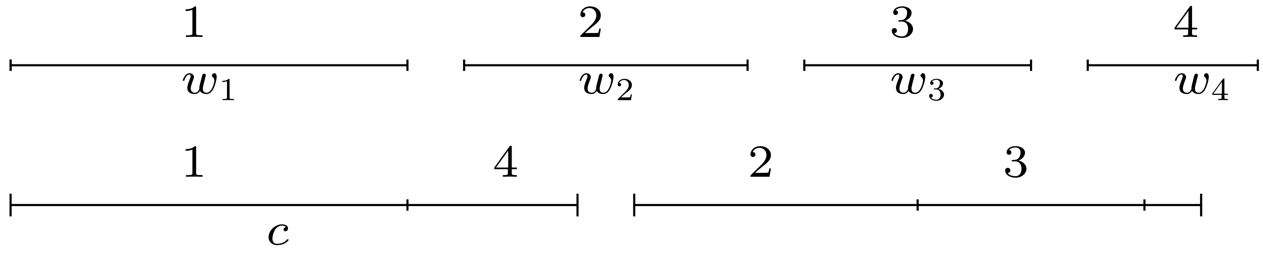
- Originally (ARCFLOW, 1999) pseudo-polynomial formulations were seen as theoretical results and rarely directly used in practice as ILP formulations (too many variables and constraints).
- Nowadays they are extremely competitive in practice. **Why?**
- 20 selected random instances ($n \in [300, 1000]$, $c \in [400, 1000]$;
  ARCFLOW: $\#$ constraints $\in [482, 1093]$, #variables $\in [32\,059, 111\,537]$);
  **8 versions of CPLEX**: number of solved instances [average CPU time]:

| Time | inst. | 6.0 (1998) | 7.0 (1999) | 8.0 (2002) | 9.0 (2003) | 10.0 (2006) | 11.0 (2007) | 12.1 (2009) | 12.6.0 (2013) |
|---|---|---|---|---|---|---|---|---|---|
| 10 minutes | 20 | 13 [366] | 10 [420] | 5 [570] | 17 [268] | 19 [162] | **20** [65] | 19 [117] | **20** [114] |
| 60 minutes | 20 | 16 [897] | 15 [1210] | 15 [2009] | **20** [343] | **20** [186] | **20** [65] | 19 [267] | **20** [114] |

# Two-Dimensional Packing Problems: Definitions

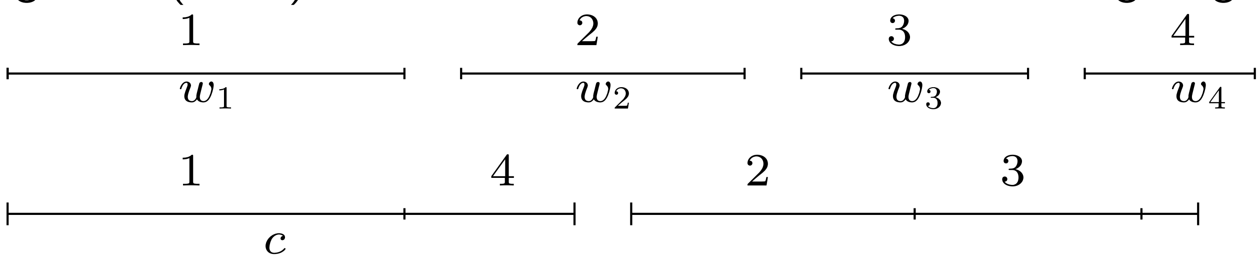
## Two-Dimensional Packing Problems: Definitions

- Geometrical interpretation of the (one-dimensional) **BPP**:
  pack a set of segments (*items*) into the minimum number of identical large segments (*bins*):



## Two-Dimensional Packing Problems: Definitions

- Geometrical interpretation of the (one-dimensional) **BPP**:
  pack a set of segments (*items*) into the minimum number of identical large segments (*bins*):




- **Two possible two-dimensional extensions.**

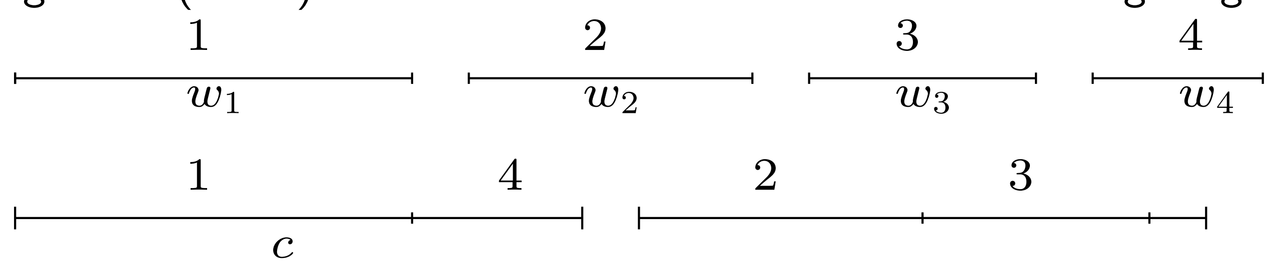
  Given $n$ rectangular *items*, each having integer **height** $h_j$ and **width** $w_j$ $(j = 1, \ldots, n)$,

---

**1) Two-Dimensional Bin Packing Problem (2BPP)**:
given an unlimited number of identical **rectangular bins** of integer **height** $H$ and **width** $W$,

## Two-Dimensional Packing Problems: Definitions

- Geometrical interpretation of the (one-dimensional) **BPP**:
  pack a set of segments (*items*) into the minimum number of identical large segments (*bins*):




- **Two possible two-dimensional extensions.**

  Given $n$ rectangular *items*, each having integer **height** $h_j$ and **width** $w_j$ $(j = 1, \ldots, n)$,

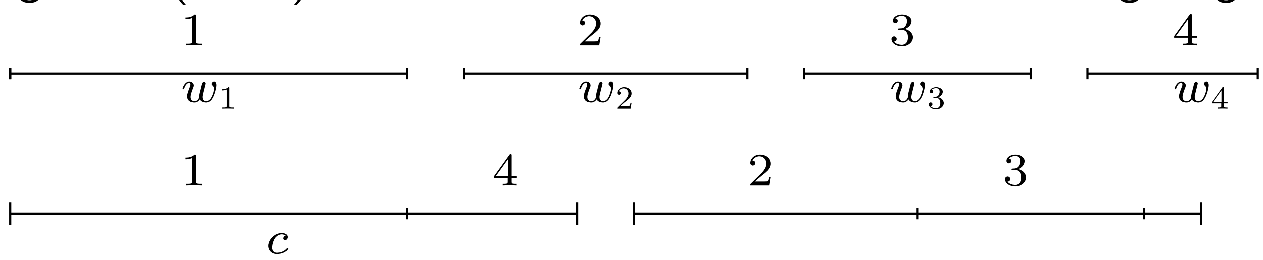
---

**1) Two-Dimensional Bin Packing Problem (2BPP)**:
given an unlimited number of identical **rectangular bins** of integer **height** $H$ and **width** $W$,
pack all the items, without overlapping, into the minimum number of bins
(find the minimum number of **cutting patterns** providing all the items).

---

# Two-Dimensional Packing Problems: Definitions

- Geometrical interpretation of the (one-dimensional) **BPP**:
  pack a set of segments (*items*) into the minimum number of identical large segments (*bins*):




- **Two possible two-dimensional extensions.**

  Given $n$ rectangular *items*, each having integer **height** $h_j$ and **width** $w_j$ $(j = 1, \dots, n)$,

---

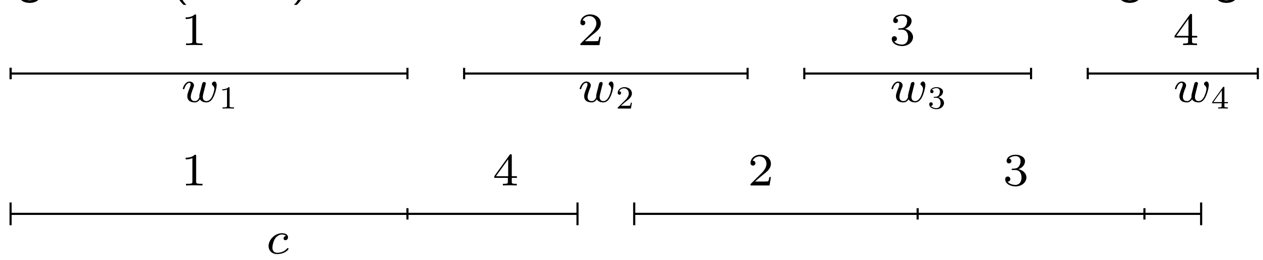
1) **Two-Dimensional Bin Packing Problem (2BPP)**:
given an unlimited number of identical **rectangular bins** of integer **height** $H$ and **width** $W$,
pack all the items, without overlapping, into the minimum number of bins
(find the minimum number of **cutting patterns** providing all the items).

---

2) **Two-Dimensional Strip Packing Problem (2SPP)**:
given a single **open-ended bin (strip)** of **width** $W$ and **infinite height**

# Two-Dimensional Packing Problems: Definitions

- Geometrical interpretation of the (one-dimensional) **BPP**:
  pack a set of segments (*items*) into the minimum number of identical large segments (*bins*):




- **Two possible two-dimensional extensions.**

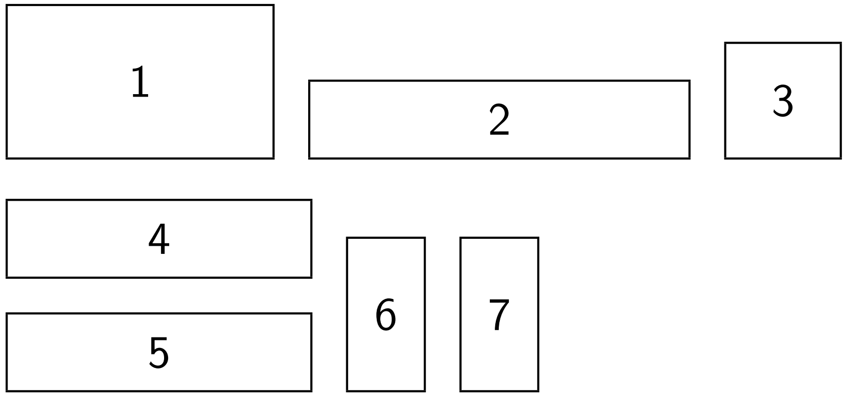
  Given $n$ rectangular *items*, each having integer **height** $h_j$ and **width** $w_j$ $(j = 1, \ldots, n)$,

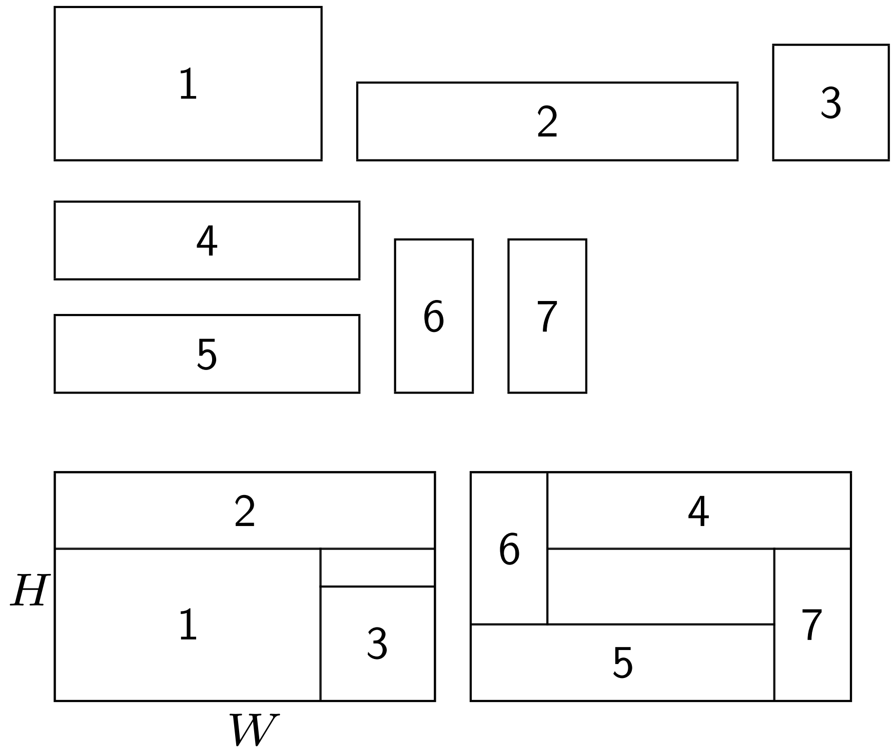
---

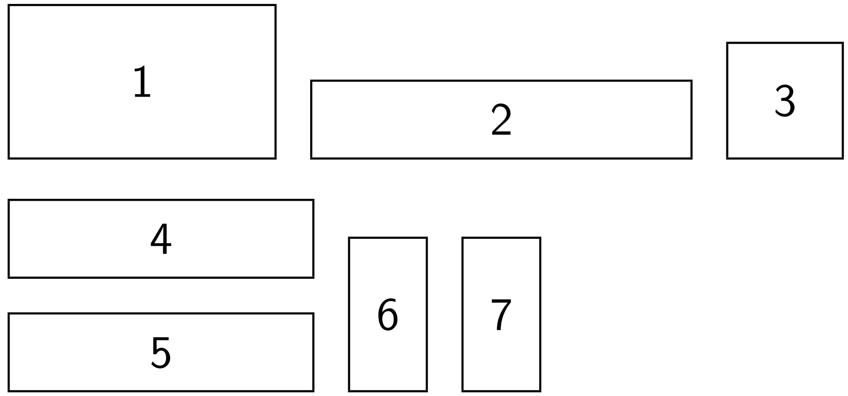
**1) Two-Dimensional Bin Packing Problem (2BPP)**:
given an unlimited number of identical **rectangular bins** of integer **height** $H$ and **width** $W$,
pack all the items, without overlapping, into the minimum number of bins
(find the minimum number of **cutting patterns** providing all the items).

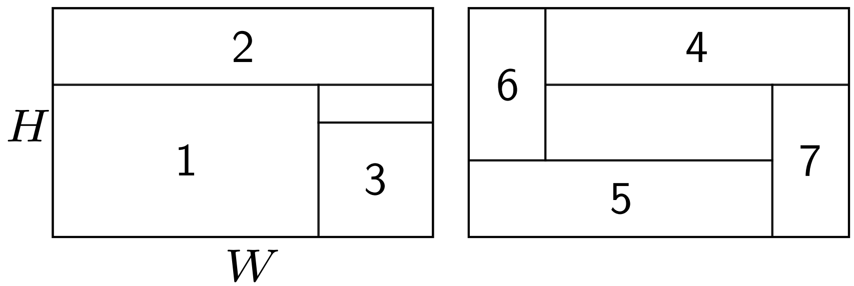
---

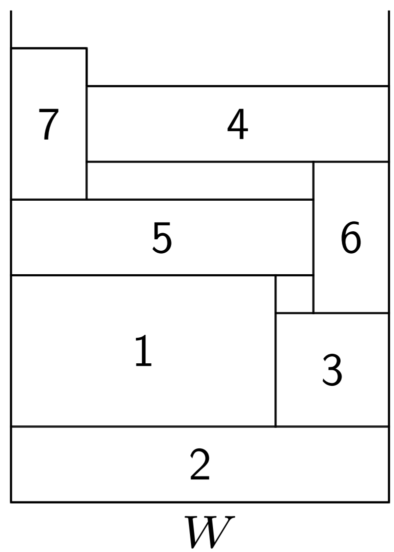
**2) Two-Dimensional Strip Packing Problem (2SPP)**:
given a single **open-ended bin (strip)** of **width** $W$ and **infinite height**
determine a cutting pattern providing all the items
such that the height to which the strip is filled is minimized.
(Also called **1.5-dimensional packing**.)


1
2
3
4
5
6
7


1
2
3
4
5
6
7
2
$H$
1
3
$W$
6
4
7
5


1
2
3
4
5
6
7


2
H
1
3
W
6
4
7
5


7
4
5
6
1
3
2
W

# Applications

# Applications

- Industrial cutting. Cutting from:
  - standardized stock pieces (glass industry, wood industry, ...) $\Longrightarrow$ 2BPP;

## Applications

- Industrial cutting. Cutting from:
  - standardized stock pieces (glass industry, wood industry, ...) $\Longrightarrow$ 2BPP;
  - $rolls$ (textile industry, paper industry, ...) $\Longrightarrow$ 2SPP;

# Applications

- Industrial cutting. Cutting from:
  - standardized stock pieces (glass industry, wood industry, ...) $\Longrightarrow$ 2BPP;
  - *rolls* (textile industry, paper industry, ...) $\Longrightarrow$ 2SPP;
- Transportation:
  - packing on floors, shelves, truck beds, ...

# Applications

- Industrial cutting. Cutting from:
  - standardized stock pieces (glass industry, wood industry, ...) $\Longrightarrow$ 2BPP;
  - $rolls$ (textile industry, paper industry, ...) $\Longrightarrow$ 2SPP;
- Transportation:
  - packing on floors, shelves, truck beds, ...
  - packing into containers (3-Dimensional Bin Packing Problem, reduction to a series of 2BPP)

# Applications

- Industrial cutting. Cutting from:
  - standardized stock pieces (glass industry, wood industry, ...) $\Longrightarrow$ 2BPP;
  - *rolls* (textile industry, paper industry, ...) $\Longrightarrow$ 2SPP;
- Transportation:
  - packing on floors, shelves, truck beds, ...
  - packing into containers (3-Dimensional Bin Packing Problem, reduction to a series of 2BPP)
- Memory sharing: shared storage multiprocessor system: 2SPP with

| | | |
|---:|:---:|:---|
| job $j$ | $\longleftrightarrow$ | rectangle $j$ |
| memory requirement | $\longleftrightarrow$ | $w_j$ (contiguous locations) |
| time requirement | $\longleftrightarrow$ | $h_j$ |
| system | $\longleftrightarrow$ | strip |
| available memory | $\longleftrightarrow$ | $W$ |
| time | $\longleftrightarrow$ | height |

## Applications

- Industrial cutting. Cutting from:
  - standardized stock pieces (glass industry, wood industry, ...) $\Longrightarrow$ 2BPP;
  - *rolls* (textile industry, paper industry, ...) $\Longrightarrow$ 2SPP;
- Transportation:
  - packing on floors, shelves, truck beds, ...
  - packing into containers (3-Dimensional Bin Packing Problem, reduction to a series of 2BPP)
- Memory sharing: shared storage multiprocessor system: 2SPP with

| | | |
|---|---|---|
| job $j$ | $\longleftrightarrow$ | rectangle $j$ |
| memory requirement | $\longleftrightarrow$ | $w_j$ (contiguous locations) |
| time requirement | $\longleftrightarrow$ | $h_j$ |
| system | $\longleftrightarrow$ | strip |
| available memory | $\longleftrightarrow$ | $W$ |
| time | $\longleftrightarrow$ | height |

## Complexity

## Applications

- Industrial cutting. Cutting from:
  - standardized stock pieces (glass industry, wood industry, ...) $\Longrightarrow$ 2BPP;
  - *rolls* (textile industry, paper industry, ...) $\Longrightarrow$ 2SPP;
- Transportation:
  - packing on floors, shelves, truck beds, ...
  - packing into containers (3-Dimensional Bin Packing Problem, reduction to a series of 2BPP)
- Memory sharing: shared storage multiprocessor system: 2SPP with

| | | |
|---|---|---|
| job $j$ | $\longleftrightarrow$ | rectangle $j$ |
| memory requirement | $\longleftrightarrow$ | $w_j$ (contiguous locations) |
| time requirement | $\longleftrightarrow$ | $h_j$ |
| system | $\longleftrightarrow$ | strip |
| available memory | $\longleftrightarrow$ | $W$ |
| time | $\longleftrightarrow$ | height |

## Complexity

- Both the 2BPP and the 2SPP are special cases of the BPP;
- both are **strongly $\mathcal{NP}$-hard**.

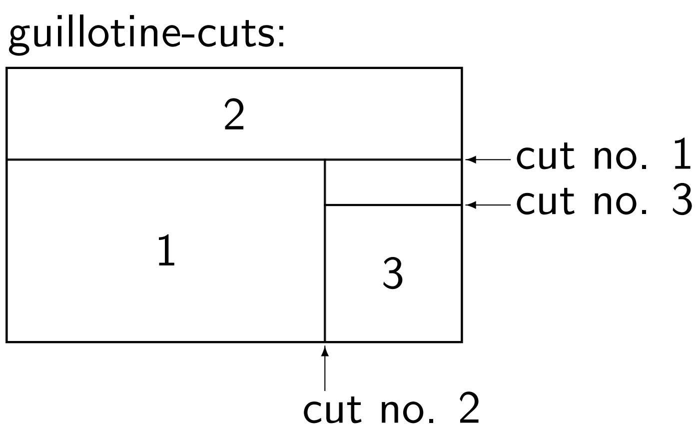
# Variants

## Variants

- **Guillotine Cuts**: In cutting applications it may be imposed that the patterns be such that the items can be obtained by sequential edge-to-edge cuts parallel to the edges of the bin.

# Variants

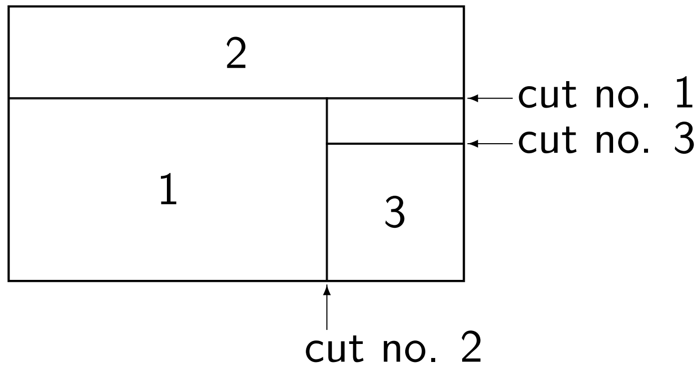
- **Guillotine Cuts**: In cutting applications it may be imposed that the patterns be such that the items can be obtained by sequential edge-to-edge cuts parallel to the edges of the bin.



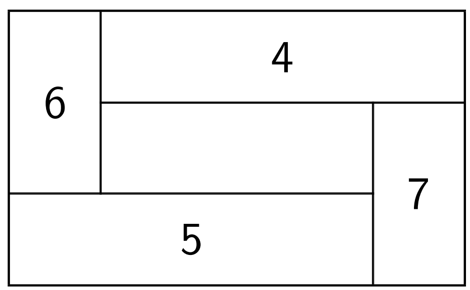
# Variants

- **Guillotine Cuts**: In cutting applications it may be imposed that the patterns be such that the items can be obtained by sequential edge-to-edge cuts parallel to the edges of the bin.

guillotine-cuts:



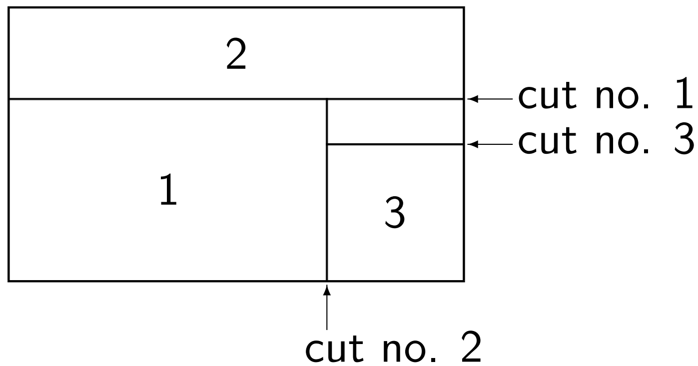

non guillotine-cuts:



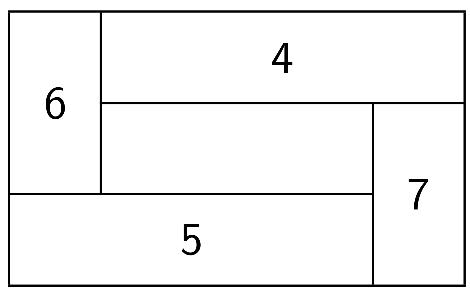
## Variants

- **Guillotine Cuts**: In cutting applications it may be imposed that the patterns be such that the items can be obtained by sequential edge-to-edge cuts parallel to the edges of the bin.

guillotine-cuts:




non guillotine-cuts:




- additional constraints: limit on the number of cuts per bin (2,3).

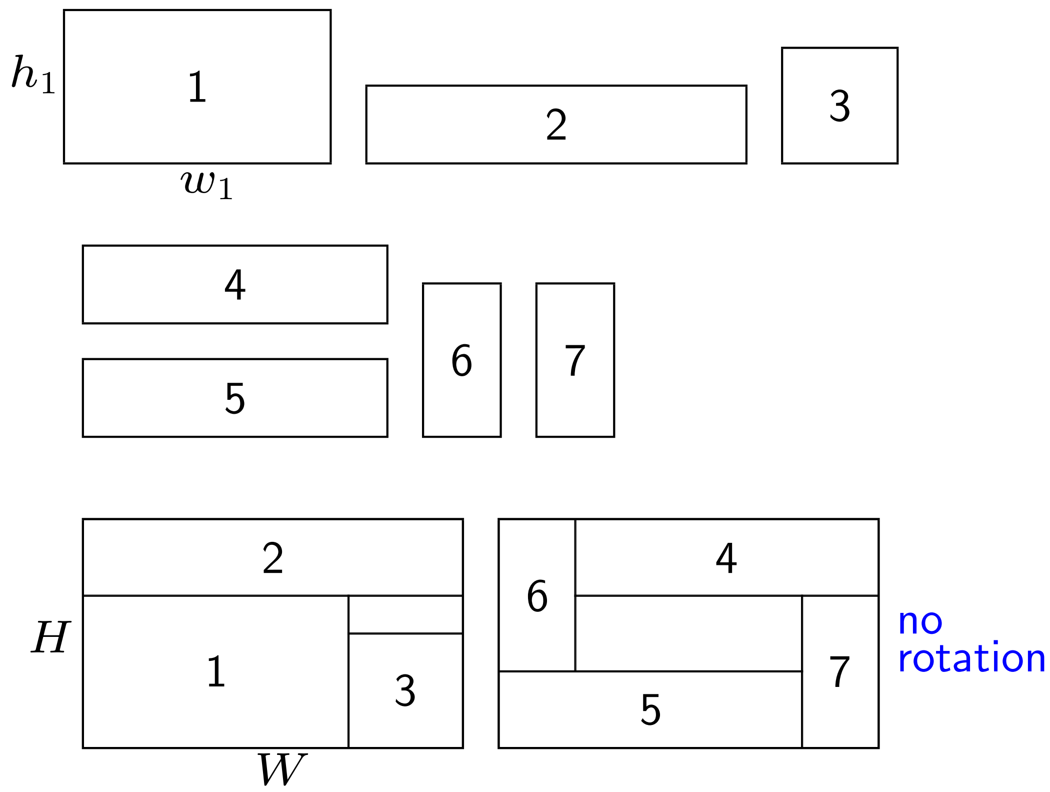
# Variants

# Variants

- **Item Rotation**: if the items in demand do not have a prefixed orientation with respect to the bins then they may be rotated (usually by $90^\circ$).

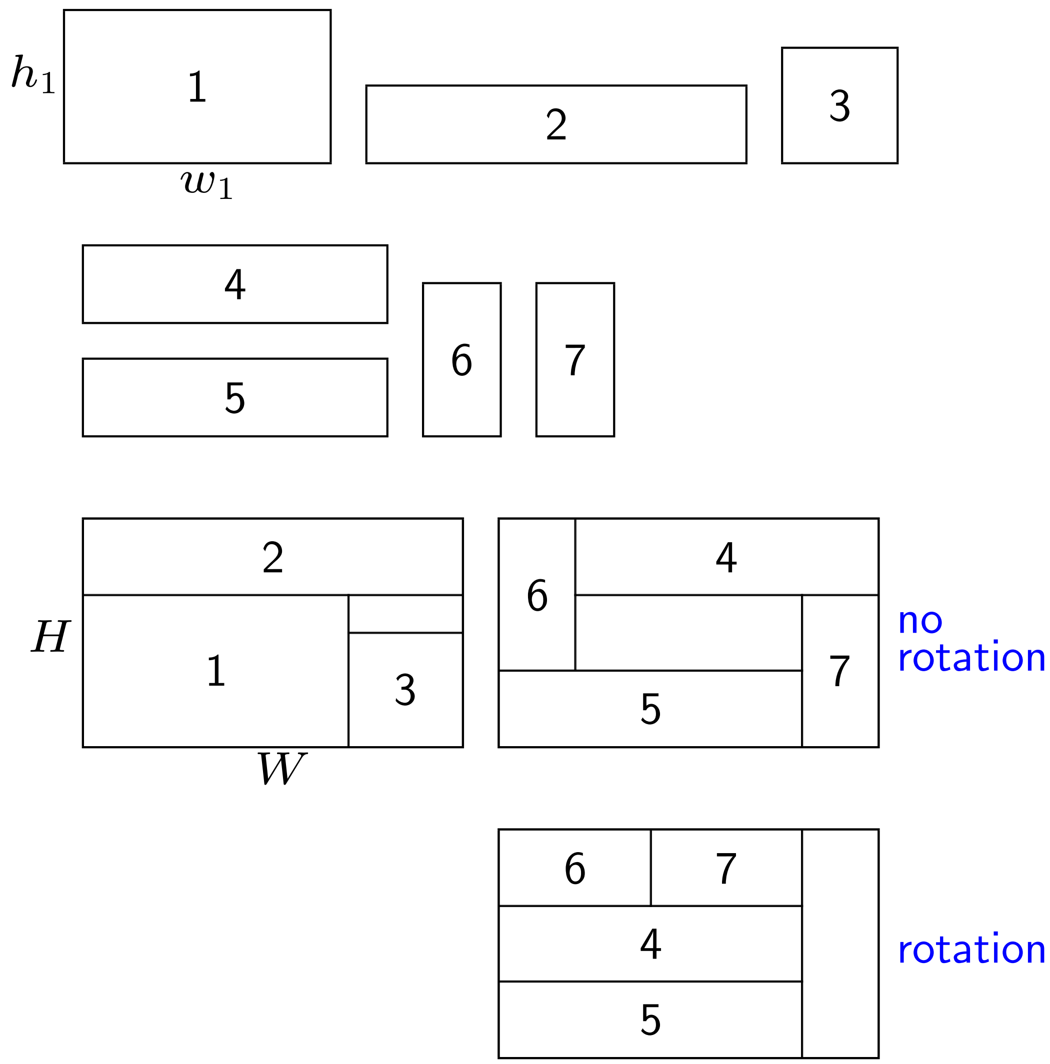
# Variants

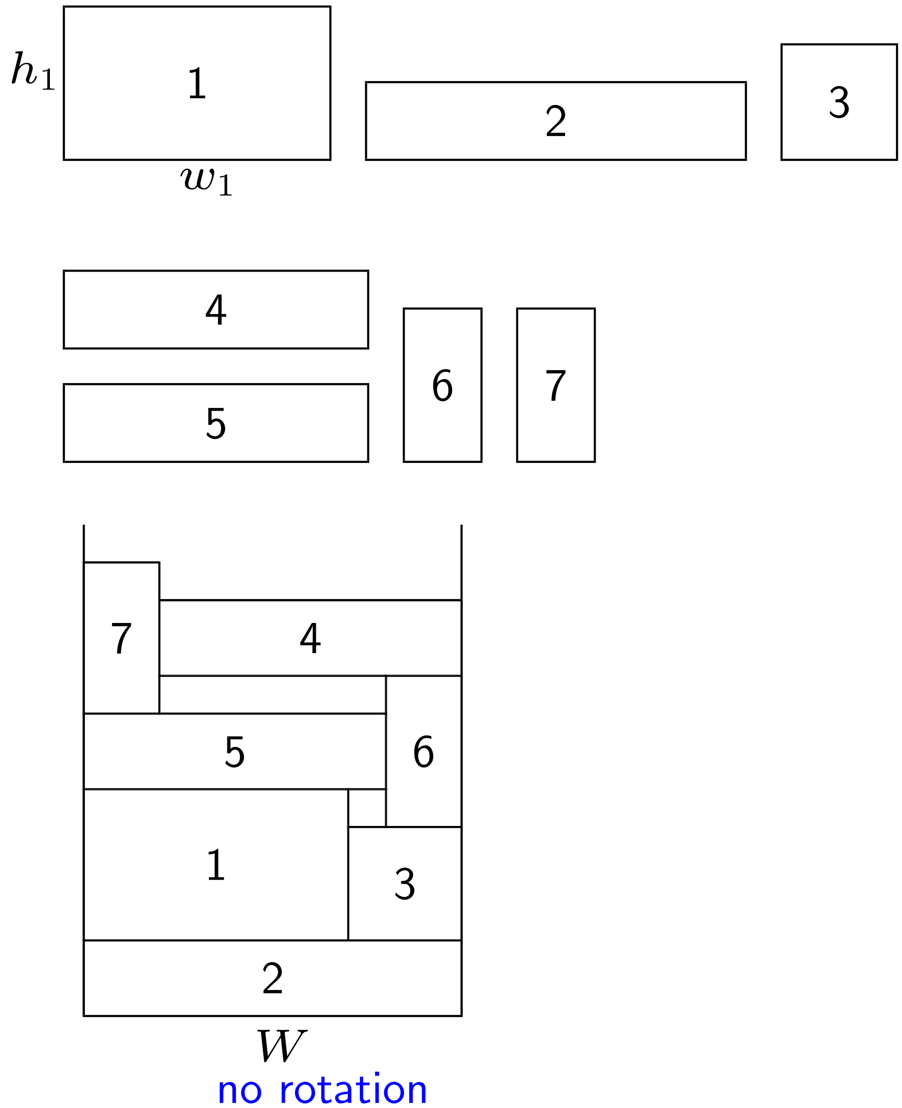
- **Item Rotation**: if the items in demand do not have a prefixed orientation with respect to the bins then they may be rotated (usually by $90^\circ$).

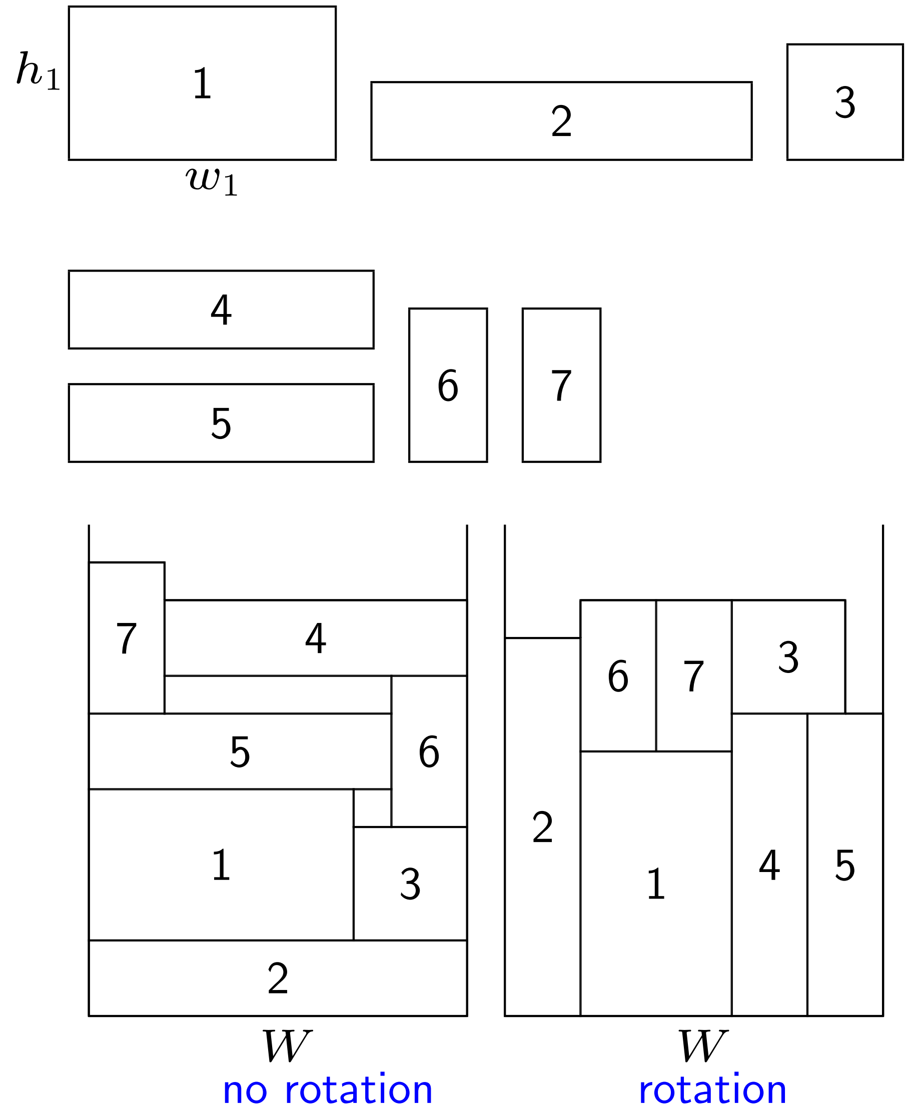


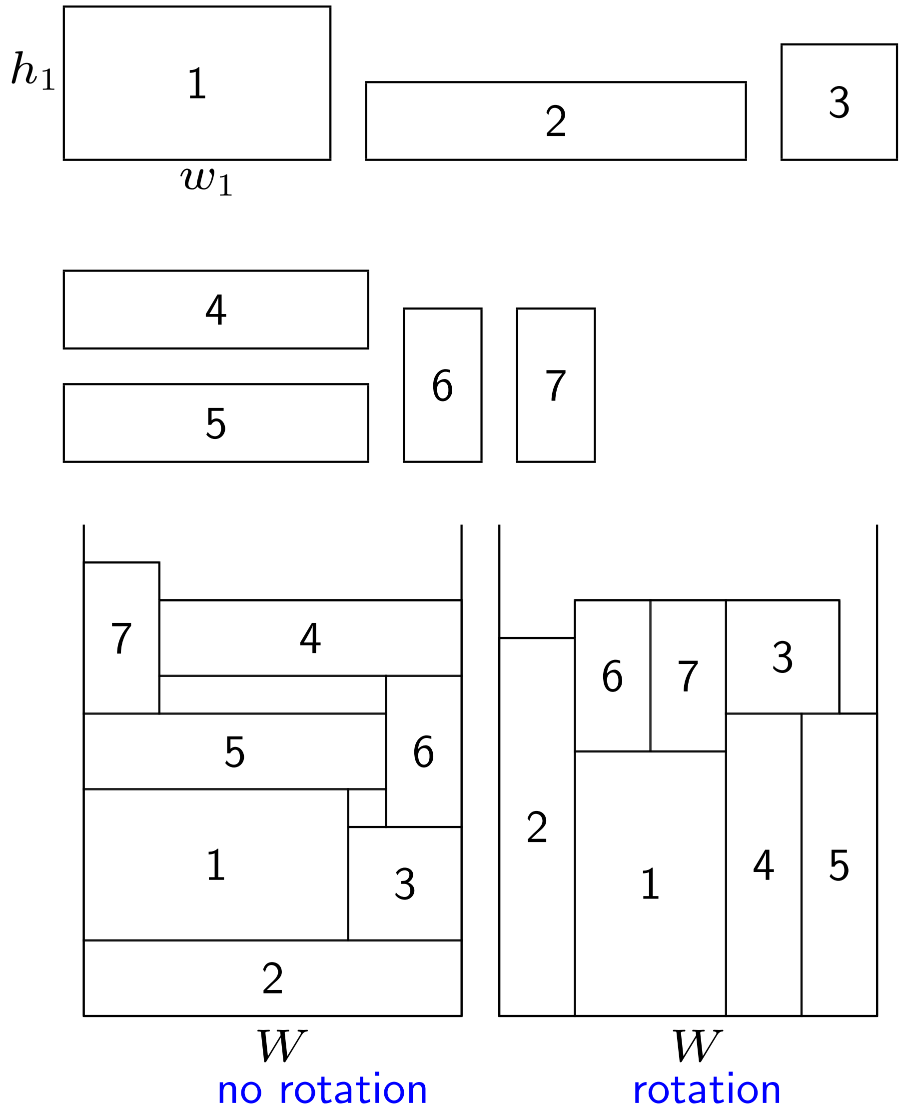
# Variants

- **Item Rotation**: if the items in demand do not have a prefixed orientation with respect to the bins then they may be rotated (usually by $90^\circ$).

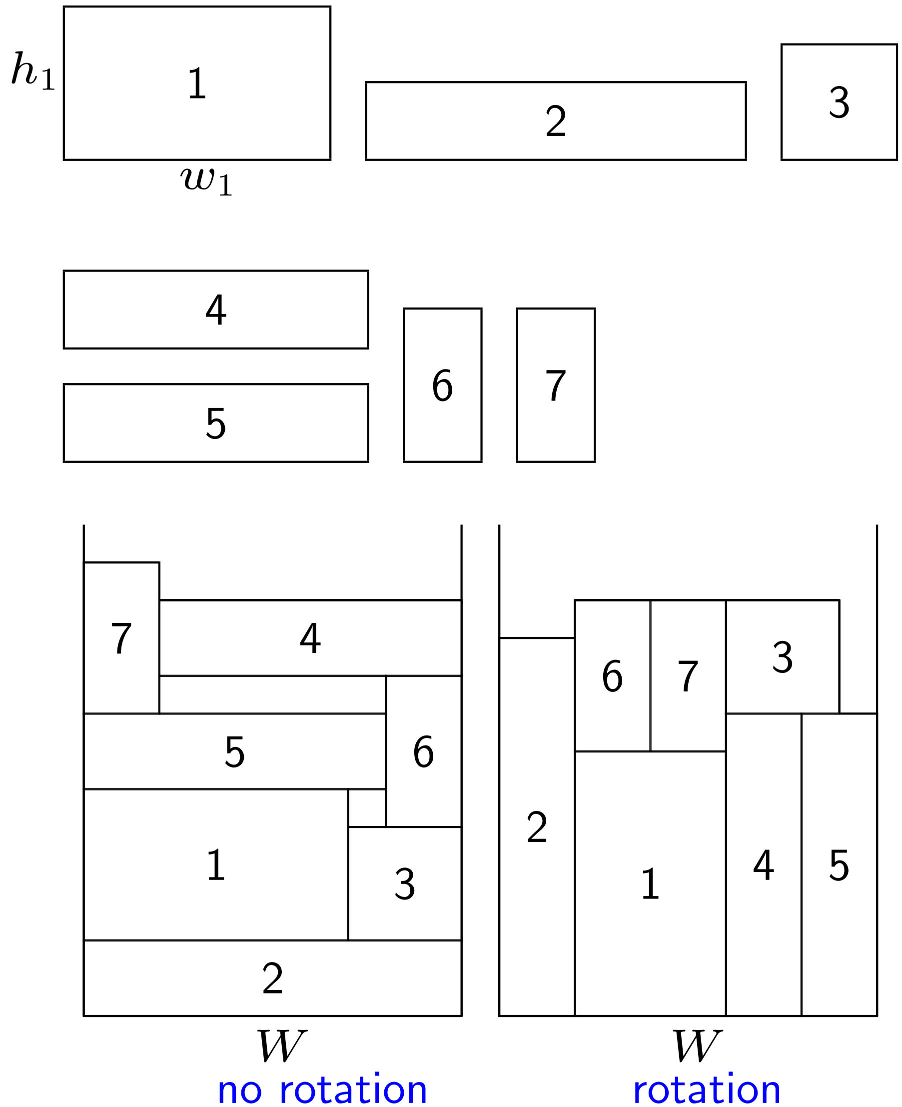



$h_1$
1
$w_1$
2
3
4
5
6
7
7
4
5
6
1
3
2
$W$
no rotation


$h_1$
1
$w_1$
2
3
4
5
6
7
7
4
5
6
1
3
2
$W$
no rotation
2
6
7
3
1
4
5
$W$
rotation




- Guillotine-cuts and rotations are frequent in other two-dimensional packing problems (Two-Dimensional Cutting Stock, Two-Dimensional Knapsack)




- Guillotine-cuts and rotations are frequent in other two-dimensional packing problems (Two-Dimensional Cutting Stock, Two-Dimensional Knapsack)
- For two-dimensional bin (strip) packing problems most results concern the case: **no guillotine-cut required**, **no rotation allowed** (implicitly assumed in the following).

# Approximation algorithms

# Approximation algorithms

Two main families of heuristic algorithms:

- **one-phase algorithms:** directly pack the items into the bins;

# Approximation algorithms

Two main families of heuristic algorithms:

- **one-phase algorithms:** directly pack the items into the bins;
- **two-phase algorithms:**
  - *Phase 1*: pack the items into a single strip;

# Approximation algorithms

Two main families of heuristic algorithms:

- **one-phase algorithms:** directly pack the items into the bins;
- **two-phase algorithms:**
  - *Phase 1*: pack the items into a single strip;
  - *Phase 2*: use the strip solution to construct a packing into bins.

# Approximation algorithms

Two main families of heuristic algorithms:

- **one-phase algorithms:** directly pack the items into the bins;
- **two-phase algorithms:**
  - *Phase 1*: pack the items into a single strip;
  - *Phase 2*: use the strip solution to construct a packing into bins.
- **Shelf algorithms:** in most of the approaches the bin/strip packing is obtained by placing the items, from left to right, in rows forming levels ($shelves$):
  - 1st shelf = bottom of the bin/strip;

## Approximation algorithms

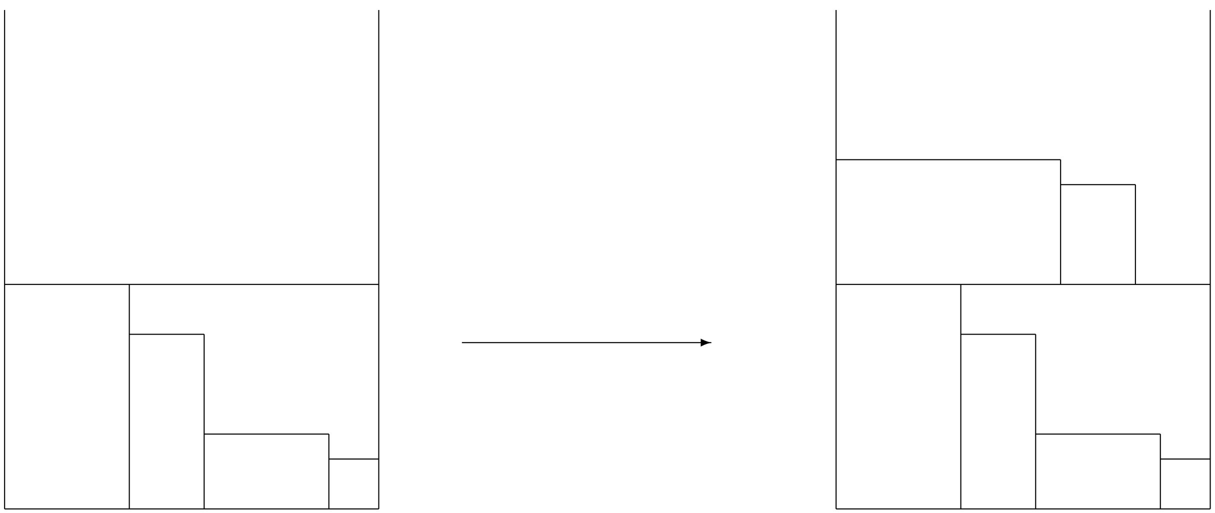
Two main families of heuristic algorithms:

- **one-phase algorithms:** directly pack the items into the bins;
- **two-phase algorithms:**
  - *Phase 1*: pack the items into a single strip;
  - *Phase 2*: use the strip solution to construct a packing into bins.
- **Shelf algorithms:** in most of the approaches the bin/strip packing is obtained by placing the items, from left to right, in rows forming levels ($shelves$):
  - 1st shelf = bottom of the bin/strip;
  - subsequent shelves = horizontal line given by the top of the tallest item in the shelf below.

# Approximation algorithms

Two main families of heuristic algorithms:

- **one-phase algorithms:** directly pack the items into the bins;
- **two-phase algorithms:**
  - *Phase 1*: pack the items into a single strip;
  - *Phase 2*: use the strip solution to construct a packing into bins.
- **Shelf algorithms:** in most of the approaches the bin/strip packing is obtained by placing the items, from left to right, in rows forming levels ($shelves$):
  - 1st shelf = bottom of the bin/strip;
  - subsequent shelves = horizontal line given by the top of the tallest item in the shelf below.



# Shelf packing strategies (2SPP)

# Shelf packing strategies (2SPP)

- sort the items by nonincreasing height (assumed in the following);
- $j$ = current item, $s$ = last created shelf:

## Shelf packing strategies (2SPP)

- sort the items by nonincreasing height (assumed in the following);
- $j$ = current item, $s$ = last created shelf:
- **Next-Fit Decreasing Height (NFDH):** pack $j$ left justified in shelf $s$, if it fits; otherwise, create a new shelf $(s+1)$, and pack $j$ left justified into it.

## Shelf packing strategies (2SPP)

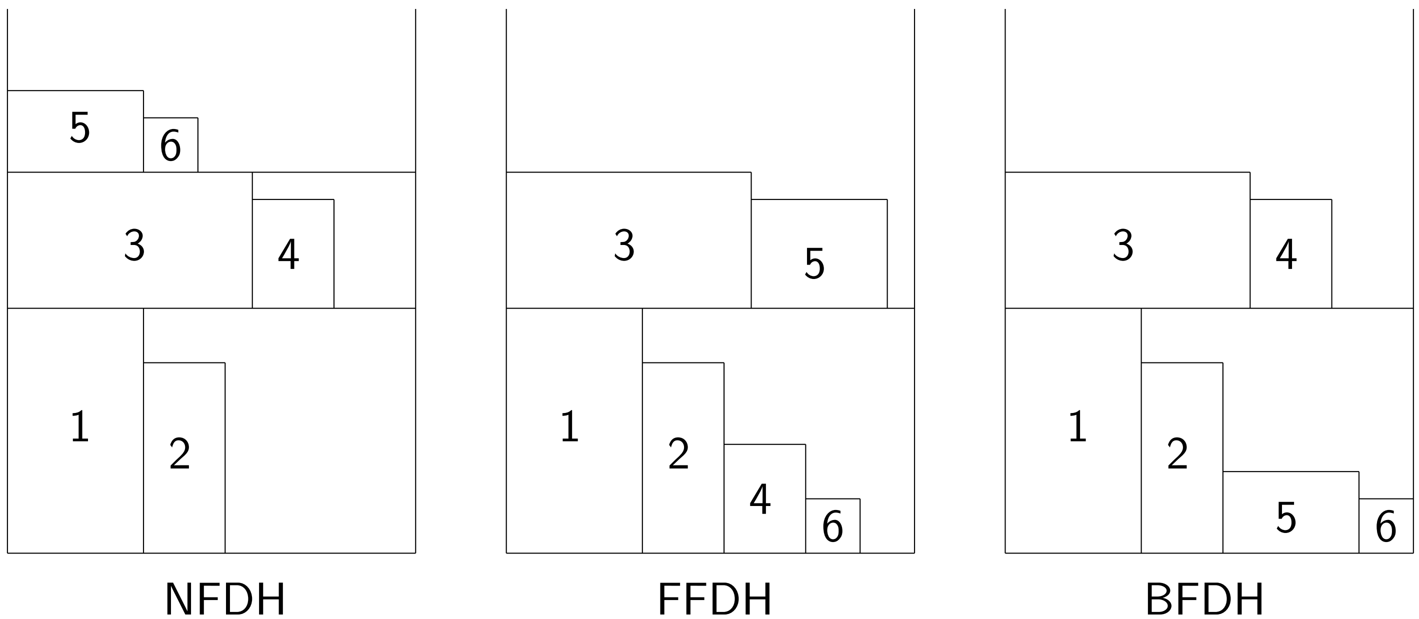
- sort the items by nonincreasing height (assumed in the following);
- $j$ = current item, $s$ = last created shelf:
- **Next-Fit Decreasing Height (NFDH):** pack $j$ left justified in shelf $s$, if it fits; otherwise, create a new shelf $(s+1)$, and pack $j$ left justified into it.
- **First-Fit Decreasing Height (FFDH):** pack $j$ left justified in the first shelf where it fits, if any; if no shelf is feasible, initialize a new shelf as in NFDH.

# Shelf packing strategies (2SPP)

- sort the items by nonincreasing height (assumed in the following);
- $j$ = current item, $s$ = last created shelf:
- **Next-Fit Decreasing Height (NFDH):** pack $j$ left justified in shelf $s$, if it fits; otherwise, create a new shelf $(s + 1)$, and pack $j$ left justified into it.
- **First-Fit Decreasing Height (FFDH):** pack $j$ left justified in the first shelf where it fits, if any; if no shelf is feasible, initialize a new shelf as in NFDH.
- **Best-Fit Decreasing Height (BFDH):** pack $j$ left justified in the feasible shelf which minimizes the unused horizontal space; if no shelf is feasible, initialize a new shelf as in NFDH.

# Shelf packing strategies (2SPP)

- sort the items by nonincreasing height (assumed in the following);
- $j$ = current item, $s$ = last created shelf:
- **Next-Fit Decreasing Height (NFDH):** pack $j$ left justified in shelf $s$, if it fits; otherwise, create a new shelf $(s + 1)$, and pack $j$ left justified into it.
- **First-Fit Decreasing Height (FFDH):** pack $j$ left justified in the first shelf where it fits, if any; if no shelf is feasible, initialize a new shelf as in NFDH.
- **Best-Fit Decreasing Height (BFDH):** pack $j$ left justified in the feasible shelf which minimizes the unused horizontal space; if no shelf is feasible, initialize a new shelf as in NFDH.



# Worst-Case Performance (2SPP)

## Worst-Case Performance (2SPP)

- If the heights are **normalized** so that $\max_j\{h_j\} = 1$, for the **Strip packing** we have (Coffman, Garey, Johnson, and Tarjan, 1980):

$$NFDH(I) \leq 2 \cdot OPT(I) + 1 \;\; \forall \; I$$

# Worst-Case Performance (2SPP)

- If the heights are **normalized** so that $\max_j\{h_j\} = 1$, for the **Strip packing** we have (Coffman, Garey, Johnson, and Tarjan, 1980):

$$NFDH(I) \leq 2 \cdot OPT(I) + 1 \;\; \forall \; I$$

and

$$FFDH(I) \leq \frac{17}{10} \cdot OPT(I) + 1 \;\; \forall \; I$$

## Worst-Case Performance (2SPP)

- If the heights are **normalized** so that $\max_j\{h_j\} = 1$, for the **Strip packing** we have (Coffman, Garey, Johnson, and Tarjan, 1980):

$$NFDH(I) \leq 2 \cdot OPT(I) + 1 \ \ \forall \ I$$

and

$$FFDH(I) \leq \frac{17}{10} \cdot OPT(I) + 1 \ \ \forall \ I$$

- Remind: for the BPP, $\overline{r}(NF) = 2$, $\overline{r}(FF) = \frac{17}{10}$.

## Worst-Case Performance (2SPP)

- If the heights are **normalized** so that $\max_j\{h_j\} = 1$, for the **Strip packing** we have (Coffman, Garey, Johnson, and Tarjan, 1980):

$$NFDH(I) \leq 2 \cdot OPT(I) + 1 \;\; \forall \; I$$

and

$$FFDH(I) \leq \frac{17}{10} \cdot OPT(I) + 1 \;\; \forall \; I$$

- Remind: for the BPP, $\overline{r}(NF) = 2$, $\overline{r}(FF) = \frac{17}{10}$.

- Both bounds are tight.
- If the $h_j$'s are **not normalized**, only the additive term is affected.

# Worst-Case Performance (2SPP)

- If the heights are **normalized** so that $\max_j\{h_j\} = 1$, for the **Strip packing** we have (Coffman, Garey, Johnson, and Tarjan, 1980):

$$NFDH(I) \le 2 \cdot OPT(I) + 1 \;\; \forall \, I$$

  and

$$FFDH(I) \le \frac{17}{10} \cdot OPT(I) + 1 \;\; \forall \, I$$

- Remind: for the BPP, $\overline{r}(NF) = 2$, $\overline{r}(FF) = \frac{17}{10}$.

- Both bounds are tight.
- If the $h_j$'s are **not normalized**, only the additive term is affected.

- Both algorithms can be implemented so as to require $O(n \log n)$ time, through the appropriate data structures used for the 1BPP.

# Two-phase algorithms (2BPP)

## Two-phase algorithms (2BPP)

- **Hybrid First-Fit (HFF,** Chung, Garey, and Johnson, 1982):
  - *Phase 1*: strip packing through **FFDH** $\rightarrow H_1, H_2, \ldots$ =heights of the resulting shelves ($H_1 \geq H_2 \ldots$ by construction).

# Two-phase algorithms (2BPP)

- **Hybrid First-Fit (HFF,** Chung, Garey, and Johnson, 1982):
  - *Phase 1*: strip packing through **FFDH** $\rightarrow H_1, H_2, \ldots$ =heights of the resulting shelves ($H_1 \geq H_2 \ldots$ by construction).
  - *Phase 2*: one-dimensional bin packing problem over the shelves:

# Two-phase algorithms (2BPP)

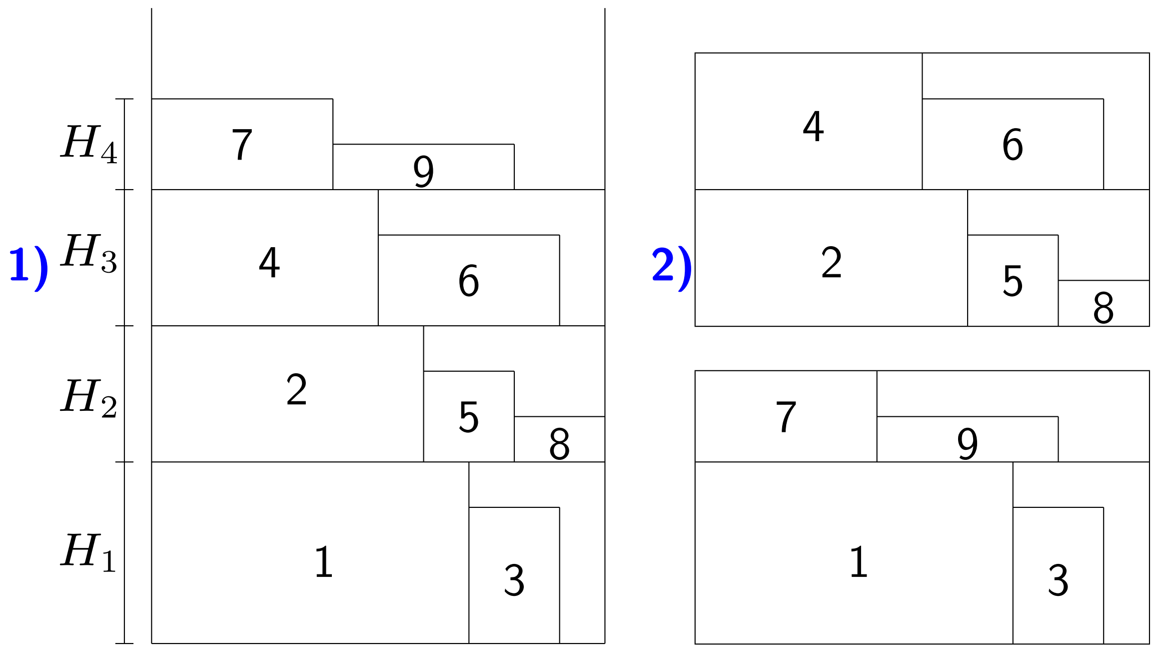
- **Hybrid First-Fit (HFF,** Chung, Garey, and Johnson, 1982):
  - *Phase 1*: strip packing through **FFDH** $\rightarrow H_1, H_2, \dots$ =heights of the resulting shelves ($H_1 \geq H_2 \dots$ by construction).
  - *Phase 2*: one-dimensional bin packing problem over the shelves:
    item sizes $H_i$, bin capacity $H$: solve through the **FFD** algorithm (BPP):
    – initialize bin 1 to pack shelf 1;

## Two-phase algorithms (2BPP)

- **Hybrid First-Fit (HFF,** Chung, Garey, and Johnson, 1982):
  - *Phase 1*: strip packing through **FFDH** $\rightarrow H_1, H_2, \ldots$ =heights of the resulting shelves ($H_1 \geq H_2 \ldots$ by construction).
  - *Phase 2*: one-dimensional bin packing problem over the shelves:
    item sizes $H_i$, bin capacity $H$: solve through the **FFD** algorithm (BPP):
    – initialize bin 1 to pack shelf 1;
    – **for** $i := 2, \ldots$ **do** pack shelf $i$ into the lowest indexed bin where it fits, if any (**otherwise** initialize a new bin).

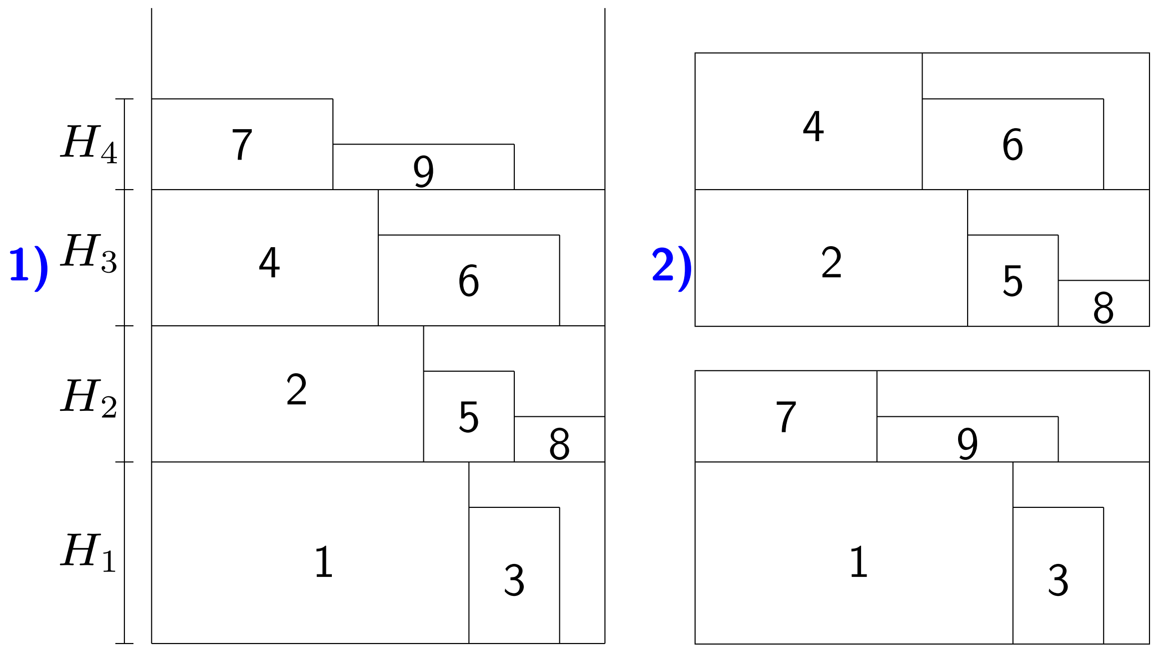
## Two-phase algorithms (2BPP)

- **Hybrid First-Fit (HFF,** Chung, Garey, and Johnson, 1982):
  - *Phase 1*: strip packing through **FFDH** $\rightarrow H_1, H_2, \ldots$ =heights of the resulting shelves ($H_1 \geq H_2 \ldots$ by construction).
  - *Phase 2*: one-dimensional bin packing problem over the shelves:
    item sizes $H_i$, bin capacity $H$: solve through the **FFD** algorithm (BPP):
    – initialize bin 1 to pack shelf 1;
    – **for** $i := 2, \ldots$ **do** pack shelf $i$ into the lowest indexed bin where it fits, if any (**otherwise** initialize a new bin).



# Two-phase algorithms (2BPP)

- **Hybrid First-Fit (HFF,** Chung, Garey, and Johnson, 1982):
  - *Phase 1*: strip packing through **FFDH** $\rightarrow H_1, H_2, \ldots$ =heights of the resulting shelves ($H_1 \geq H_2 \ldots$ by construction).
  - *Phase 2*: one-dimensional bin packing problem over the shelves:
    item sizes $H_i$, bin capacity $H$: solve through the **FFD** algorithm (BPP):
    – initialize bin 1 to pack shelf 1;
    – **for** $i := 2, \ldots$ **do** pack shelf $i$ into the lowest indexed bin where it fits, if any (**otherwise** initialize a new bin).




- If the heights are **normalized to 1**, $HFF(I) \leq \frac{17}{8} \cdot OPT(I) + 5 \;\; \forall \; I$

# Other two-phase algorithms (2BPP)

## Other two-phase algorithms (2BPP)

- **Hybrid Best-Fit (HBF,** Berkey and Wang, 1982):
  - *Phase 1*: strip packing through the BFDH strategy;
  - *Phase 2*: 1BPP solved through the $Best\text{-}Fit\ Decreasing$ algorithm.

# Other two-phase algorithms (2BPP)

- **Hybrid Best-Fit (HBF,** Berkey and Wang, 1982):
  - *Phase 1*: strip packing through the BFDH strategy;
  - *Phase 2*: 1BPP solved through the *Best-Fit Decreasing* algorithm.
- **Hybrid Next-Fit (HNF,** Frenk and Galambos, 1987):
  - *Phase 1*: strip packing through the NFDH strategy;
  - *Phase 2*: 1BPP solved through the *Next-Fit Decreasing* algorithm.

# Other two-phase algorithms (2BPP)

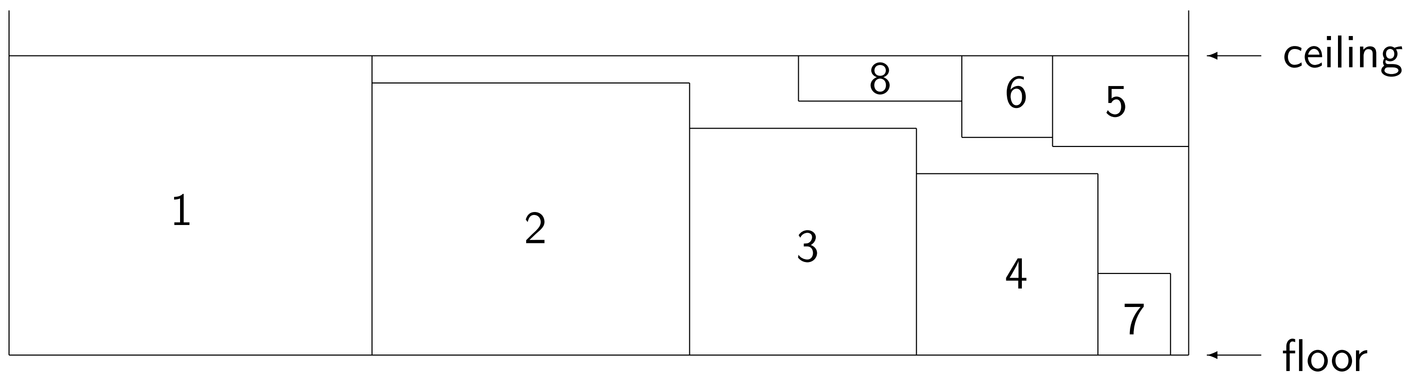
- **Hybrid Best-Fit (HBF,** Berkey and Wang, 1982):
  - *Phase 1*: strip packing through the BFDH strategy;
  - *Phase 2*: 1BPP solved through the *Best-Fit Decreasing* algorithm.
- **Hybrid Next-Fit (HNF,** Frenk and Galambos, 1987):
  - *Phase 1*: strip packing through the NFDH strategy;
  - *Phase 2*: 1BPP solved through the *Next-Fit Decreasing* algorithm.
- Both $O(n \log n)$ time.

## Other two-phase algorithms (2BPP)

- **Hybrid Best-Fit (HBF,** Berkey and Wang, 1982):
  - *Phase 1*: strip packing through the BFDH strategy;
  - *Phase 2*: 1BPP solved through the $Best\text{-}Fit\ Decreasing$ algorithm.
- **Hybrid Next-Fit (HNF,** Frenk and Galambos, 1987):
  - *Phase 1*: strip packing through the NFDH strategy;
  - *Phase 2*: 1BPP solved through the $Next\text{-}Fit\ Decreasing$ algorithm.
- Both $O(n \log n)$ time.
- **Floor-Ceiling (FC,** Lodi, Martello, and Vigo, 2000):
  - *ceiling* $=$ horizontal line defined by the top edge of the tallest item packed in the shelf;
  - pack on the shelf floor (left to right) and with the top edge on the ceiling (right to left).
  - $O(n^3)$ time but better experimental performance.

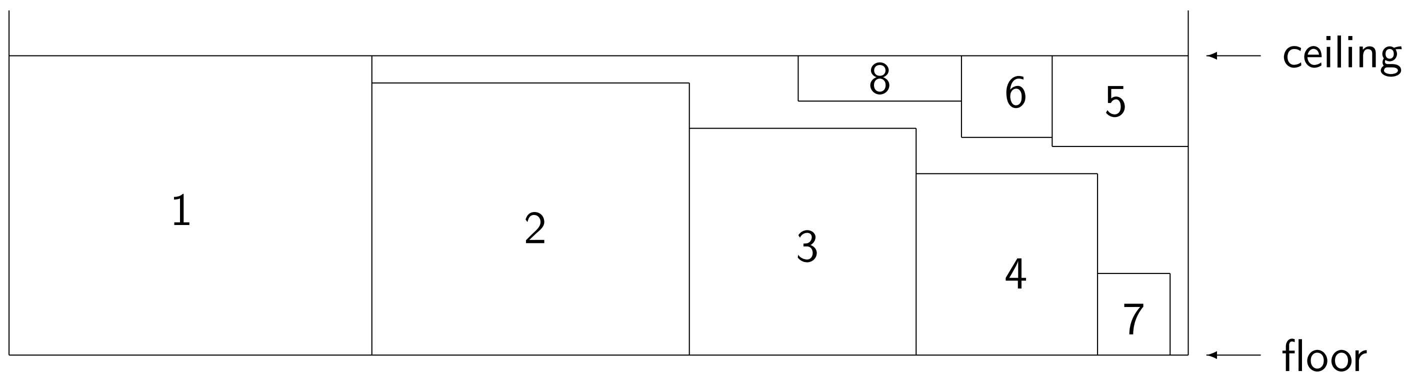
# Other two-phase algorithms (2BPP)

- **Hybrid Best-Fit (HBF,** Berkey and Wang, 1982):
  - *Phase 1*: strip packing through the BFDH strategy;
  - *Phase 2*: 1BPP solved through the *Best-Fit Decreasing* algorithm.
- **Hybrid Next-Fit (HNF,** Frenk and Galambos, 1987):
  - *Phase 1*: strip packing through the NFDH strategy;
  - *Phase 2*: 1BPP solved through the *Next-Fit Decreasing* algorithm.
- Both $O(n \log n)$ time.
- **Floor-Ceiling (FC,** Lodi, Martello, and Vigo, 2000):
  - *ceiling* = horizontal line defined by the top edge of the tallest item packed in the shelf;
  - pack on the shelf floor (left to right) and with the top edge on the ceiling (right to left).
  - $O(n^3)$ time but better experimental performance.



# Other two-phase algorithms (2BPP)

- **Hybrid Best-Fit (HBF,** Berkey and Wang, 1982):
  - *Phase 1*: strip packing through the BFDH strategy;
  - *Phase 2*: 1BPP solved through the *Best-Fit Decreasing* algorithm.
- **Hybrid Next-Fit (HNF,** Frenk and Galambos, 1987):
  - *Phase 1*: strip packing through the NFDH strategy;
  - *Phase 2*: 1BPP solved through the *Next-Fit Decreasing* algorithm.
- Both $O(n \log n)$ time.
- **Floor-Ceiling (FC,** Lodi, Martello, and Vigo, 2000):
  - *ceiling* $=$ horizontal line defined by the top edge of the tallest item packed in the shelf;
  - pack on the shelf floor (left to right) and with the top edge on the ceiling (right to left).
  - $O(n^3)$ time but better experimental performance.




- **Knapsack packing** (Lodi, Martello, and Vigo, 1999): optimize the packing on the shelves by solving associated knapsack problems ($\mathcal{NP}$-hard).

# One-phase algorithms (2BPP)

## One-phase algorithms (2BPP)

- **Finite Next-Fit (FNF)**:
  - pack the current item in the current shelf of the current bin, if it fits;

# One-phase algorithms (2BPP)

- **Finite Next-Fit (FNF)**:
  - pack the current item in the current shelf of the current bin, if it fits;
  - otherwise, create a new (current) shelf

# One-phase algorithms (2BPP)

- **Finite Next-Fit (FNF)**:
  - pack the current item in the current shelf of the current bin, if it fits;
  - otherwise, create a new (current) shelf
    either in the current bin (if enough vertical space is available)
    or by initializing a new bin.

# One-phase algorithms (2BPP)

- **Finite Next-Fit (FNF)**:
  - pack the current item in the current shelf of the current bin, if it fits;
  - otherwise, create a new (current) shelf

    either in the current bin (if enough vertical space is available)

    or by initializing a new bin.
- **Finite First-Fit (FFF)**:
  - pack the current item in the lowest shelf of the first bin where it fits;

## One-phase algorithms (2BPP)

- **Finite Next-Fit (FNF)**:
  - pack the current item in the current shelf of the current bin, if it fits;
  - otherwise, create a new (current) shelf

    either in the current bin (if enough vertical space is available)

    or by initializing a new bin.
- **Finite First-Fit (FFF)**:
  - pack the current item in the lowest shelf of the first bin where it fits;
  - if no shelf can accommodate it, create a new shelf

# One-phase algorithms (2BPP)

- **Finite Next-Fit (FNF)**:
  - pack the current item in the current shelf of the current bin, if it fits;
  - otherwise, create a new (current) shelf

    either in the current bin (if enough vertical space is available)

    or by initializing a new bin.
- **Finite First-Fit (FFF)**:
  - pack the current item in the lowest shelf of the first bin where it fits;
  - if no shelf can accommodate it, create a new shelf

    either in the first suitable bin

    or by initializing a new bin

# One-phase algorithms (2BPP)

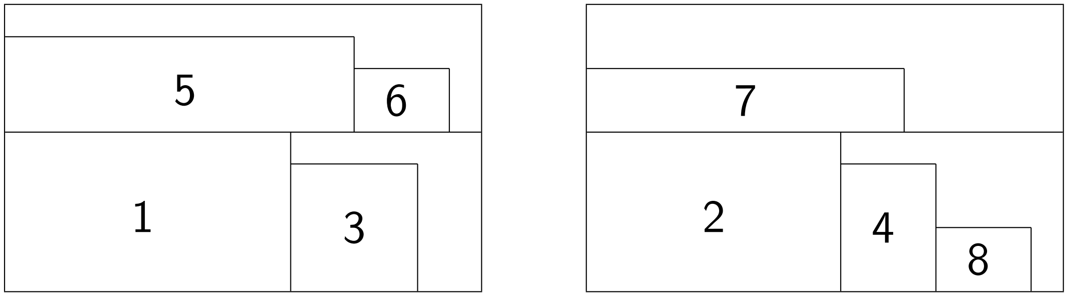
- **Finite Next-Fit (FNF)**:
  - pack the current item in the current shelf of the current bin, if it fits;
  - otherwise, create a new (current) shelf

    either in the current bin (if enough vertical space is available)

    or by initializing a new bin.
- **Finite First-Fit (FFF)**:
  - pack the current item in the lowest shelf of the first bin where it fits;
  - if no shelf can accommodate it, create a new shelf

    either in the first suitable bin

    or by initializing a new bin



# One-phase algorithms (2BPP)

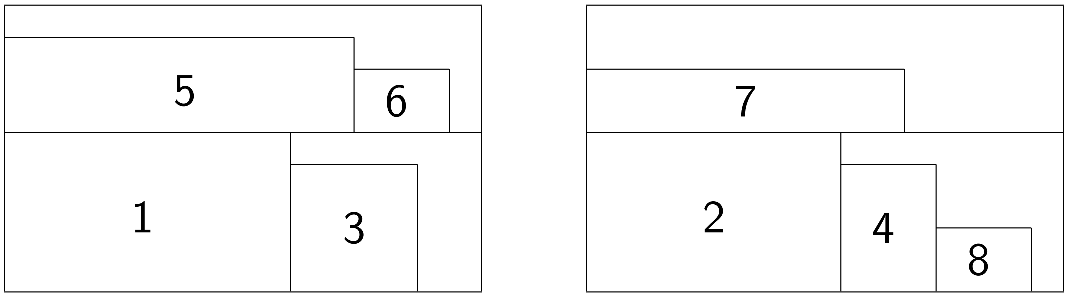
- **Finite Next-Fit (FNF)**:
  - pack the current item in the current shelf of the current bin, if it fits;
  - otherwise, create a new (current) shelf

    either in the current bin (if enough vertical space is available)

    or by initializing a new bin.
- **Finite First-Fit (FFF)**:
  - pack the current item in the lowest shelf of the first bin where it fits;
  - if no shelf can accommodate it, create a new shelf

    either in the first suitable bin

    or by initializing a new bin




- Both $O(n \log n)$ time (Berkey and Wang, 1982).

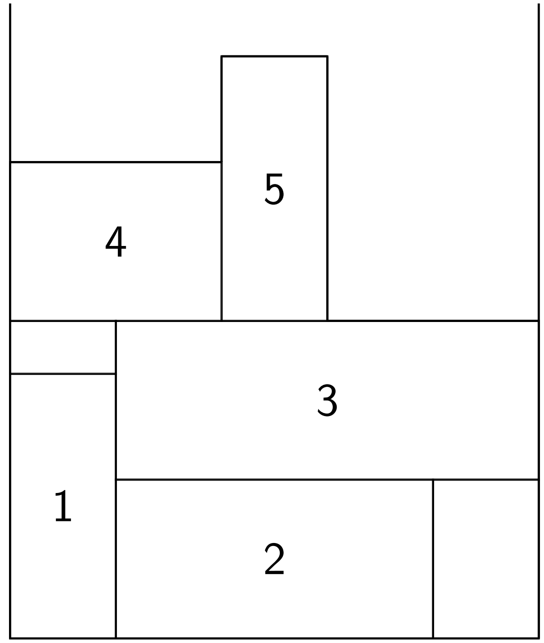
# One-phase algorithms (2BPP and 2SPP)

# One-phase algorithms (2BPP and 2SPP)

- Main non-shelf strategy:
  **Bottom-Left (BL)**: pack the current item in the lowest possible position, left justified.

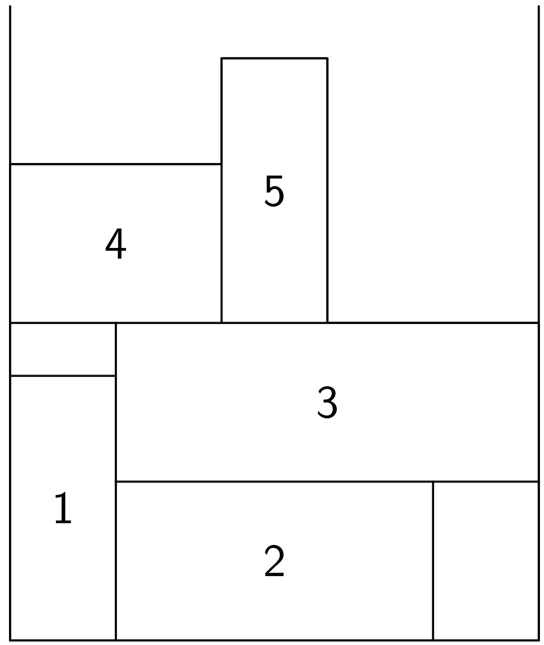
# One-phase algorithms (2BPP and 2SPP)

- Main non-shelf strategy:
  **Bottom-Left (BL)**: pack the current item in the lowest possible position, left justified.



# One-phase algorithms (2BPP and 2SPP)

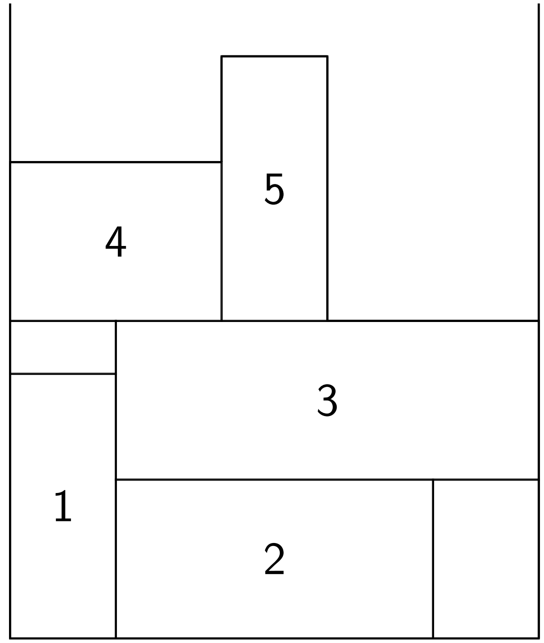
- Main non-shelf strategy:
  **Bottom-Left (BL)**: pack the current item in the lowest possible position, left justified.




- Complicated $O(n^2)$ time implementation (Chazelle).

# One-phase algorithms (2BPP and 2SPP)

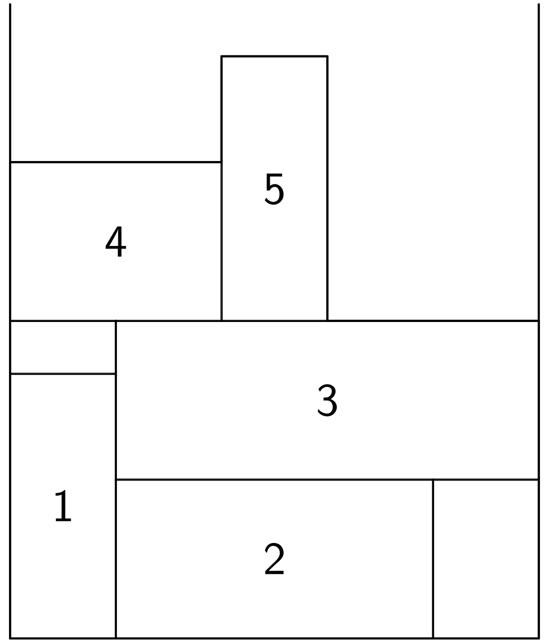
- Main non-shelf strategy:
  **Bottom-Left (BL)**: pack the current item in the lowest possible position, left justified.




- Complicated $O(n^2)$ time implementation (Chazelle).
- Worst-case performance for the 2SPP (Baker, Coffman, and Rivest, 1980):
  - if no item ordering is used, then BL may be arbitrarily bad;

# One-phase algorithms (2BPP and 2SPP)

- Main non-shelf strategy:
  **Bottom-Left (BL)**: pack the current item in the lowest possible position, left justified.




- Complicated $O(n^2)$ time implementation (Chazelle).
- Worst-case performance for the 2SPP (Baker, Coffman, and Rivest, 1980):
  - if no item ordering is used, then BL may be arbitrarily bad;
  - if the items are sorted by nonincreasing width, then

$$BL(I) \leq 3 \cdot OPT(I) \ \ \forall \ I \text{ (tight)}$$

# Approximation algorithms and schemes

# Approximation algorithms and schemes

- Mostly theoretical relevance.
- **Asymptotic approximability**:
  - First asymptotic fully polynomial-time approximation scheme for the **2SPP**: Kenyon and Remila (2000).

# Approximation algorithms and schemes

- Mostly theoretical relevance.
- **Asymptotic approximability**:
  - First asymptotic fully polynomial-time approximation scheme for the **2SPP**: Kenyon and Remila (2000).
  - Asymptotic fully polynomial-time approximation scheme for a restricted version of the **2BPP**: Caprara, Lodi and Monaci (2002).

# Approximation algorithms and schemes

- Mostly theoretical relevance.
- **Asymptotic approximability**:
  - First asymptotic fully polynomial-time approximation scheme for the **2SPP**: Kenyon and Remila (2000).
  - Asymptotic fully polynomial-time approximation scheme for a restricted version of the **2BPP**: Caprara, Lodi and Monaci (2002).
  - Bansal and Sviridenko (2004): No asymptotic polynomial time approximation scheme (APTAS) can exist for the **2BPP** unless $\mathcal{P} = \mathcal{NP}$.

## Approximation algorithms and schemes

- Mostly theoretical relevance.
- **Asymptotic approximability**:
  - First asymptotic fully polynomial-time approximation scheme for the **2SPP**: Kenyon and Remila (2000).
  - Asymptotic fully polynomial-time approximation scheme for a restricted version of the **2BPP**: Caprara, Lodi and Monaci (2002).
  - Bansal and Sviridenko (2004): No asymptotic polynomial time approximation scheme (APTAS) can exist for the **2BPP** unless $\mathcal{P} = \mathcal{NP}$.
  - Best result: General framework for approximation algorithms: asymptotic approximation guarantees arbitrarily close to 1.525 for the **2BPP** (Bansal, Caprara and Sviridenko,2006)

# Approximation algorithms and schemes

- Mostly theoretical relevance.
- **Asymptotic approximability**:
  - First asymptotic fully polynomial-time approximation scheme for the **2SPP**: Kenyon and Remila (2000).
  - Asymptotic fully polynomial-time approximation scheme for a restricted version of the **2BPP**: Caprara, Lodi and Monaci (2002).
  - Bansal and Sviridenko (2004): No asymptotic polynomial time approximation scheme (APTAS) can exist for the **2BPP** unless $\mathcal{P} = \mathcal{NP}$.
  - Best result: General framework for approximation algorithms: asymptotic approximation guarantees arbitrarily close to 1.525 for the **2BPP** (Bansal, Caprara and Sviridenko,2006)
- **Absolute approximability**:

# Approximation algorithms and schemes

- Mostly theoretical relevance.
- **Asymptotic approximability**:
  - First asymptotic fully polynomial-time approximation scheme for the **2SPP**: Kenyon and Remila (2000).
  - Asymptotic fully polynomial-time approximation scheme for a restricted version of the **2BPP**: Caprara, Lodi and Monaci (2002).
  - Bansal and Sviridenko (2004): No asymptotic polynomial time approximation scheme (APTAS) can exist for the **2BPP** unless $\mathcal{P} = \mathcal{NP}$.
  - Best result: General framework for approximation algorithms: asymptotic approximation guarantees arbitrarily close to 1.525 for the **2BPP** (Bansal, Caprara and Sviridenko,2006)
- **Absolute approximability**:
  - Zhang (2005): 3-approximation algorithm for the **2BPP**;

## Approximation algorithms and schemes

- Mostly theoretical relevance.
- **Asymptotic approximability**:
  - First asymptotic fully polynomial-time approximation scheme for the **2SPP**: Kenyon and Remila (2000).
  - Asymptotic fully polynomial-time approximation scheme for a restricted version of the **2BPP**: Caprara, Lodi and Monaci (2002).
  - Bansal and Sviridenko (2004): No asymptotic polynomial time approximation scheme (APTAS) can exist for the **2BPP** unless $\mathcal{P} = \mathcal{NP}$.
  - Best result: General framework for approximation algorithms: asymptotic approximation guarantees arbitrarily close to 1.525 for the **2BPP** (Bansal, Caprara and Sviridenko,2006)
- **Absolute approximability**:
  - Zhang (2005): 3-approximation algorithm for the **2BPP**;
  - Harren and van Stee (2009): **2-approximation algorithm for the 2BPP**; best possible polynomial time approximation for **2BPP**, unless $\mathcal{P} = \mathcal{NP}$.

## Approximation algorithms and schemes

- Mostly theoretical relevance.
- **Asymptotic approximability**:
  - First asymptotic fully polynomial-time approximation scheme for the **2SPP**: Kenyon and Remila (2000).
  - Asymptotic fully polynomial-time approximation scheme for a restricted version of the **2BPP**: Caprara, Lodi and Monaci (2002).
  - Bansal and Sviridenko (2004): No asymptotic polynomial time approximation scheme (APTAS) can exist for the **2BPP** unless $\mathcal{P} = \mathcal{NP}$.
  - Best result: General framework for approximation algorithms: asymptotic approximation guarantees arbitrarily close to 1.525 for the **2BPP** (Bansal, Caprara and Sviridenko,2006)
- **Absolute approximability**:
  - Zhang (2005): 3-approximation algorithm for the **2BPP**;
  - Harren and van Stee (2009): **2-approximation algorithm for the 2BPP**; best possible polynomial time approximation for **2BPP**, unless $\mathcal{P} = \mathcal{NP}$.
  - Harren, Jansen, Prädel, and van Stee (2014): $\frac{5}{3} + \varepsilon$-approximation algorithm for the **2SPP**.

# Lower bounds

# Lower bounds

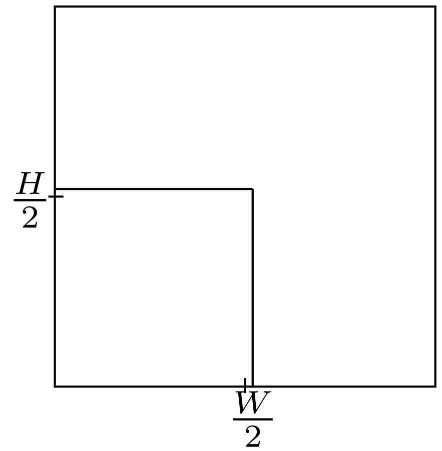
## Continuous Lower Bound

- **2BPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{HW} \right\rceil$;

# Lower bounds

## Continuous Lower Bound

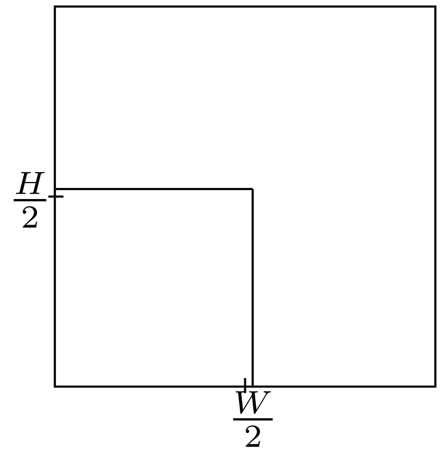
- **2BPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{HW} \right\rceil$;
  - $L_0(I) \geq \frac{1}{4} \cdot OPT(I) \ \ \forall \ I$. Tight:



# Lower bounds

## Continuous Lower Bound

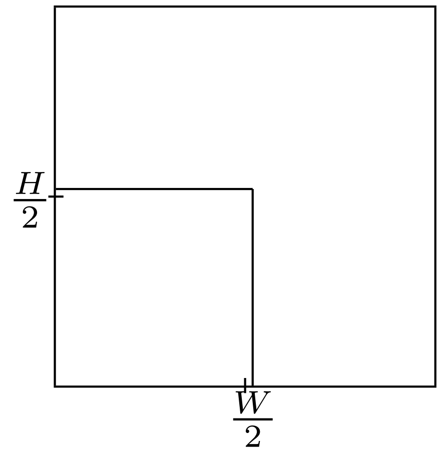
- **2BPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{HW} \right\rceil$;
  - $L_0(I) \geq \frac{1}{4} \cdot OPT(I) \quad \forall\, I$. Tight:




- **2SPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{W} \right\rceil$;

## Continuous Lower Bound

- **2BPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{HW} \right\rceil$;
  - $L_0(I) \geq \frac{1}{4} \cdot OPT(I) \ \ \forall\, I$. Tight:

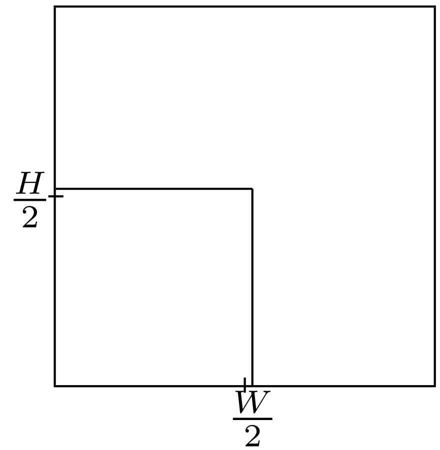



- **2SPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{W} \right\rceil$;
  - Arbitrarily bad! ($n = 1$, $w_1 = 1$ $h_1 = W$: $L_0 = 1$, $z = W$);

# Lower bounds

## Continuous Lower Bound

- **2BPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{HW} \right\rceil$;
  - $L_0(I) \geq \frac{1}{4} \cdot OPT(I) \;\; \forall\, I$. Tight:

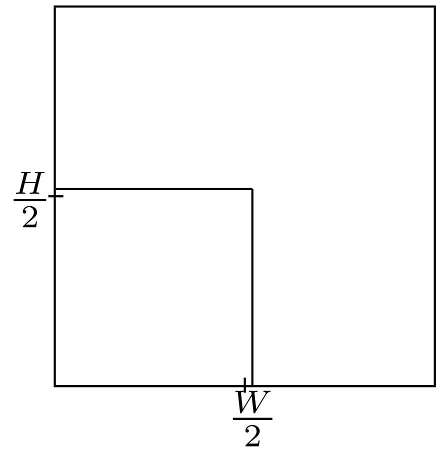



- **2SPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{W} \right\rceil$;
  - Arbitrarily bad! ($n = 1$, $w_1 = 1$ $h_1 = W$: $L_0 = 1$, $z = W$);
  - better bound: $\overline{L_0} = \max(L_0, \max_{j=1,\ldots,n}\{h_j\})$;

# Lower bounds

## Continuous Lower Bound

- **2BPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{HW} \right\rceil$;
  - $L_0(I) \geq \frac{1}{4} \cdot OPT(I) \ \ \forall\ I$. Tight:

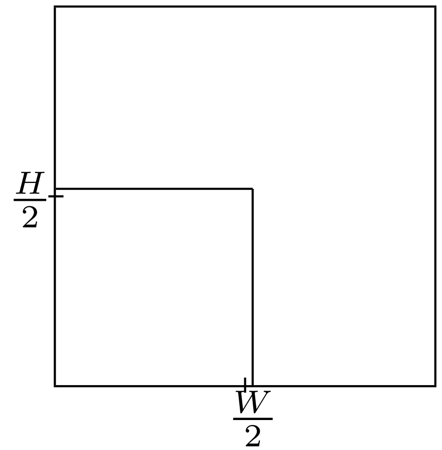



- **2SPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{W} \right\rceil$;
  - Arbitrarily bad! ($n = 1$, $w_1 = 1$ $h_1 = W$: $L_0 = 1$, $z = W$);
  - better bound: $\overline{L}_0 = \max(L_0, \max_{j=1,\ldots,n}\{h_j\})$;
  - $\overline{L}_0(I) \geq \frac{1}{2} \cdot OPT(I) \ \ \forall\ I$ (Tight) (Lodi, Martello, Monaci, Vigo 2003).

# Lower bounds

## Continuous Lower Bound

- **2BPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{HW} \right\rceil$;
  - $L_0(I) \geq \frac{1}{4} \cdot OPT(I) \;\; \forall\, I$. Tight:




- **2SPP:** $L_0 = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{W} \right\rceil$;
  - Arbitrarily bad! $(n = 1,\ w_1 = 1\ h_1 = W\colon\ L_0 = 1,\ z = W)$;
  - better bound: $\overline{L}_0 = \max(L_0, \max_{j=1,\dots,n}\{h_j\})$;
  - $\overline{L}_0(I) \geq \frac{1}{2} \cdot OPT(I) \;\; \forall\, I$ (Tight) (Lodi, Martello, Monaci, Vigo 2003).
- Other lower bounds derived from the from the (one-dimensional) BPP.

# Exact Algorithms

# Exact Algorithms

**2BPP**:

- ***Nested*** **branch-and-bound algorithm** (Martello & Vigo, 1998). Depth-first strategy:

# Exact Algorithms

**2BPP**:

- ***Nested* branch-and-bound algorithm** (Martello & Vigo, 1998). Depth-first strategy:
  - *outer tree* (from the 1BPP): items assigned to bins without specifying their actual position: at level $k$, item $k$ is assigned, in turn, to all active bins and, possibly, to a new bin;

# Exact Algorithms

**2BPP**:

- ***Nested* branch-and-bound algorithm** (Martello & Vigo, 1998). Depth-first strategy:
  - *outer tree* (from the 1BPP): items assigned to bins without specifying their actual position:
    at level $k$, item $k$ is assigned, in turn, to all active bins and, possibly, to a new bin;
  - *inner tree*: find a a feasible packing (if any) for the items assigned to the bin through
    * approximation algorithms (the packing is feasible if $z = 1$);

# Exact Algorithms

**2BPP**:

- ***Nested* branch-and-bound algorithm** (Martello & Vigo, 1998). Depth-first strategy:
  - *outer tree* (from the 1BPP): items assigned to bins without specifying their actual position: at level $k$, item $k$ is assigned, in turn, to all active bins and, possibly, to a new bin;
  - *inner tree*: find a a feasible packing (if any) for the items assigned to the bin through
    * approximation algorithms (the packing is feasible if $z = 1$);
    * lower bounds (no packing exists if $LB > 1$); if these fail,

# Exact Algorithms

**2BPP**:

- ***Nested* branch-and-bound algorithm** (Martello & Vigo, 1998). Depth-first strategy:
  - *outer tree* (from the 1BPP): items assigned to bins without specifying their actual position: at level $k$, item $k$ is assigned, in turn, to all active bins and, possibly, to a new bin;
  - *inner tree*: find a a feasible packing (if any) for the items assigned to the bin through
    * approximation algorithms (the packing is feasible if $z = 1$);
    * lower bounds (no packing exists if $LB > 1$); if these fail,
    * enumeration of all possible patterns.

# Exact Algorithms

**2BPP**:

- ***Nested* branch-and-bound algorithm** (Martello & Vigo, 1998). Depth-first strategy:
  - *outer tree* (from the 1BPP): items assigned to bins without specifying their actual position: at level $k$, item $k$ is assigned, in turn, to all active bins and, possibly, to a new bin;
  - *inner tree*: find a a feasible packing (if any) for the items assigned to the bin through
    - approximation algorithms (the packing is feasible if $z = 1$);
    - lower bounds (no packing exists if $LB > 1$); if these fail,
    - enumeration of all possible patterns.
- **Branch-and-price algorithms:**
  Pisinger and Sigurd (2007): decomposition + constraint programming;

# Exact Algorithms

**2BPP**:

- ***Nested* branch-and-bound algorithm** (Martello & Vigo, 1998). Depth-first strategy:
  - *outer tree* (from the 1BPP): items assigned to bins without specifying their actual position: at level $k$, item $k$ is assigned, in turn, to all active bins and, possibly, to a new bin;
  - *inner tree*: find a a feasible packing (if any) for the items assigned to the bin through
    - approximation algorithms (the packing is feasible if $z = 1$);
    - lower bounds (no packing exists if $LB > 1$); if these fail,
    - enumeration of all possible patterns.
- **Branch-and-price algorithms:**
  Pisinger and Sigurd (2007): decomposition + constraint programming;
- Enumerative approach for the **single bin 2BPP** (Fekete, Schepers, and van der Veen (2007));

# Exact Algorithms

**2BPP**:

- ***Nested* branch-and-bound algorithm** (Martello & Vigo, 1998). Depth-first strategy:
  - *outer tree* (from the 1BPP): items assigned to bins without specifying their actual position: at level $k$, item $k$ is assigned, in turn, to all active bins and, possibly, to a new bin;
  - *inner tree*: find a a feasible packing (if any) for the items assigned to the bin through
    * approximation algorithms (the packing is feasible if $z = 1$);
    * lower bounds (no packing exists if $LB > 1$); if these fail,
    * enumeration of all possible patterns.
- **Branch-and-price algorithms:**
  Pisinger and Sigurd (2007): decomposition + constraint programming;
- Enumerative approach for the **single bin 2BPP** (Fekete, Schepers, and van der Veen (2007));

**2SPP** :

- **Branch-and-bound algorithm:** (Martello, Monaci, and Vigo (2003));
  improvements by Boschetti and Montaletti (2010).

# Exact Algorithms

**2BPP**:

- ***Nested* branch-and-bound algorithm** (Martello & Vigo, 1998). Depth-first strategy:
  - *outer tree* (from the 1BPP): items assigned to bins without specifying their actual position: at level $k$, item $k$ is assigned, in turn, to all active bins and, possibly, to a new bin;
  - *inner tree*: find a a feasible packing (if any) for the items assigned to the bin through
    - $*$ approximation algorithms (the packing is feasible if $z = 1$);
    - $*$ lower bounds (no packing exists if $LB > 1$); if these fail,
    - $*$ enumeration of all possible patterns.
- **Branch-and-price algorithms:**
  Pisinger and Sigurd (2007): decomposition + constraint programming;
- Enumerative approach for the **single bin 2BPP** (Fekete, Schepers, and van der Veen (2007));

**2SPP** :

- **Branch-and-bound algorithm:** (Martello, Monaci, and Vigo (2003));
  improvements by Boschetti and Montaletti (2010).
- **2SPP with 90$^\circ$ item rotation** (*Stock Cutting Problem*):
  - branch-and-bound algorithm (Arahori, Imamichi, and Nagamochi (2012);

# Exact Algorithms

**2BPP**:

- ***Nested* branch-and-bound algorithm** (Martello & Vigo, 1998). Depth-first strategy:
  - *outer tree* (from the 1BPP): items assigned to bins without specifying their actual position: at level $k$, item $k$ is assigned, in turn, to all active bins and, possibly, to a new bin;
  - *inner tree*: find a a feasible packing (if any) for the items assigned to the bin through
    - approximation algorithms (the packing is feasible if $z = 1$);
    - lower bounds (no packing exists if $LB > 1$); if these fail,
    - enumeration of all possible patterns.
- **Branch-and-price algorithms:** Pisinger and Sigurd (2007): decomposition + constraint programming;
- Enumerative approach for the **single bin 2BPP** (Fekete, Schepers, and van der Veen (2007));

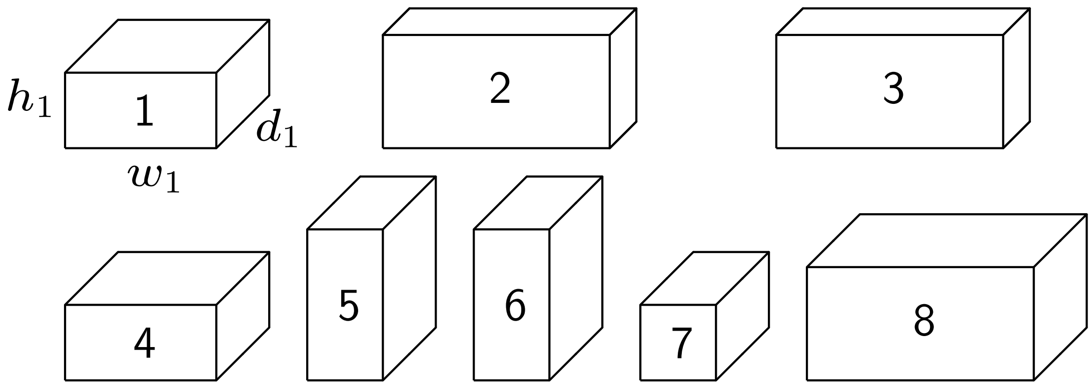
**2SPP** :

- **Branch-and-bound algorithm:** (Martello, Monaci, and Vigo (2003)); improvements by Boschetti and Montaletti (2010).
- **2SPP with 90° item rotation** (*Stock Cutting Problem*):
  - branch-and-bound algorithm (Arahori, Imamichi, and Nagamochi (2012);
  - Benders' decomposition (Delorme, Iori, and Martello (2017).

# Three-dimensional packing problems (brief outline)

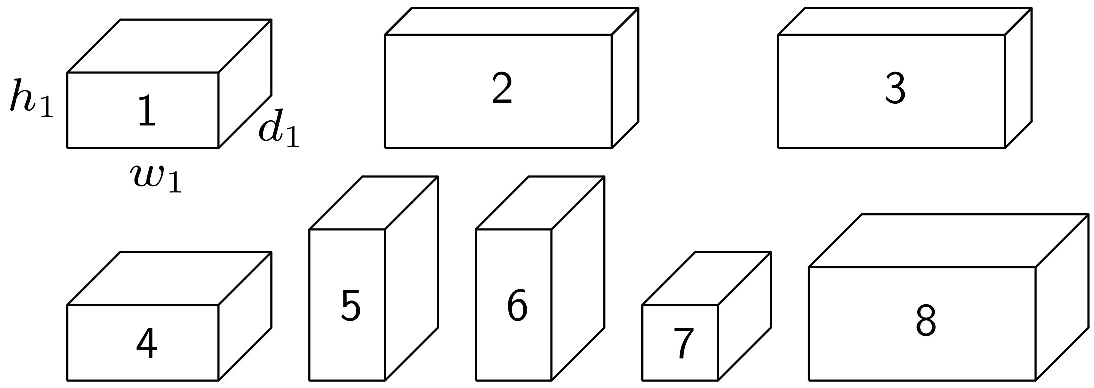
# Three-dimensional packing problems (brief outline)

- Given $n$ rectangular-shaped *boxes* with integer **height** $h_j$**,** **width** $w_j$, and **depth** $d_j$ ...



# Three-dimensional packing problems (brief outline)

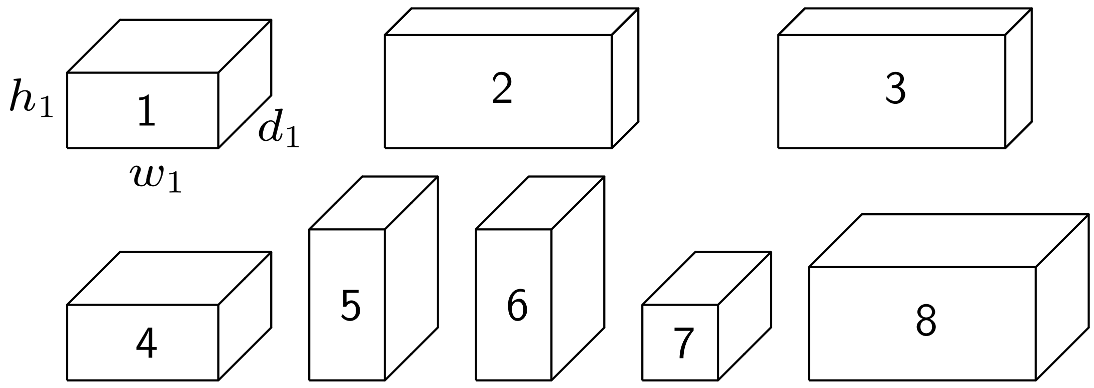
- Given $n$ rectangular-shaped *boxes* with integer **height** $h_j$**,** **width** $w_j$, and **depth** $d_j$ ...



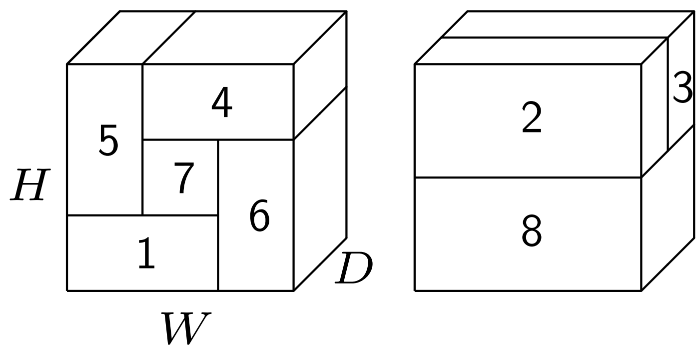

- ... and an unlimited number of identical **rectangular 3-dimensional bins** having **height** $H$**,** **width** $W$ and **depth** $D$, orthogonally pack all the boxes into the minimum number of bins (**Three-Dimensional Bin Packing Problem, 3BPP**):

## Three-dimensional packing problems (brief outline)

- Given $n$ rectangular-shaped *boxes* with integer **height** $h_j$**,** **width** $w_j$, and **depth** $d_j$ ...

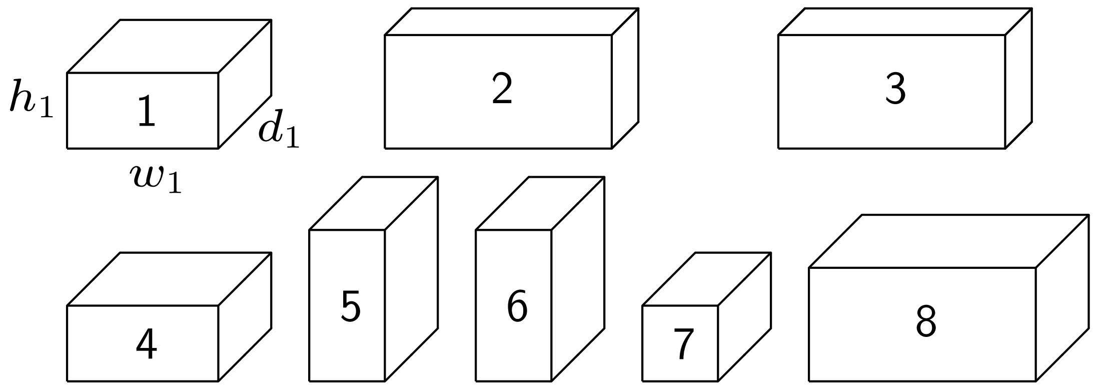



- ... and an unlimited number of identical **rectangular 3-dimensional bins** having **height** $H$**,** **width** $W$ and **depth** $D$, orthogonally pack all the boxes into the minimum number of bins (**Three-Dimensional Bin Packing Problem, 3BPP**):

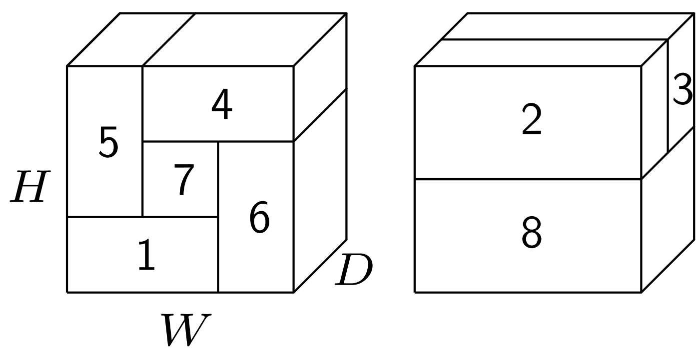


# Three-dimensional packing problems (brief outline)

- Given $n$ rectangular-shaped *boxes* with integer **height** $h_j$**, width** $w_j$, and **depth** $d_j$ ...




- ... and an unlimited number of identical **rectangular 3-dimensional bins** having **height** $H$**, width** $W$ and **depth** $D$, orthogonally pack all the boxes into the minimum number of bins (**Three-Dimensional Bin Packing Problem, 3BPP**):




- ... and a single open-ended strip of **width** $W$**, depth** $D$, and **infinite height**, orthogonally pack all the boxes by minimizing the height to which the strip is filled (**Three-Dimensional Strip Packing Problem, 3SPP**).

# Three-dimensional packing problems (brief outline)

# Three-dimensional packing problems (brief outline)

- **Complexity**: obviously NP-hard in the strong sense.

# Three-dimensional packing problems (brief outline)

- **Complexity**: obviously NP-hard in the strong sense.
- **Applications:**
  - *Loading*: containers, vehicles (trucks, freight cars), pallets;

# Three-dimensional packing problems (brief outline)

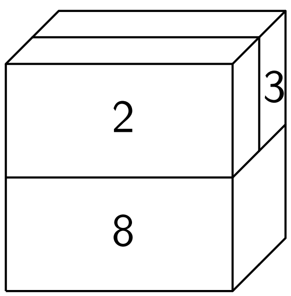
- **Complexity**: obviously NP-hard in the strong sense.
- **Applications:**
  - *Loading*: containers, vehicles (trucks, freight cars), pallets;
  - *Packaging design*: boxes, cases;

# Three-dimensional packing problems (brief outline)

- **Complexity**: obviously NP-hard in the strong sense.
- **Applications:**
  - *Loading*: containers, vehicles (trucks, freight cars), pallets;
  - *Packaging design*: boxes, cases;
  - *Cutting*: foam rubber (arm-chair production).

## Three-dimensional packing problems (brief outline)

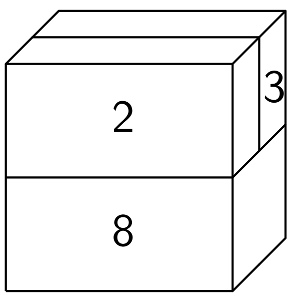
- **Complexity**: obviously NP-hard in the strong sense.
- **Applications:**
  - *Loading*: containers, vehicles (trucks, freight cars), pallets;
  - *Packaging design*: boxes, cases;
  - *Cutting*: foam rubber (arm-chair production).
- **Variants and additional constraints:**
  - Guillotine cuts:



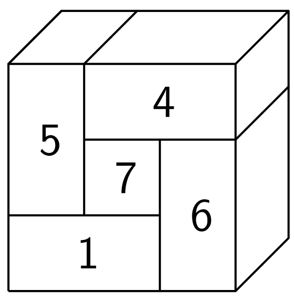

guillotine-cuts

# Three-dimensional packing problems (brief outline)

- **Complexity**: obviously NP-hard in the strong sense.
- **Applications:**
  - *Loading*: containers, vehicles (trucks, freight cars), pallets;
  - *Packaging design*: boxes, cases;
  - *Cutting*: foam rubber (arm-chair production).
- **Variants and additional constraints:**
  - Guillotine cuts:



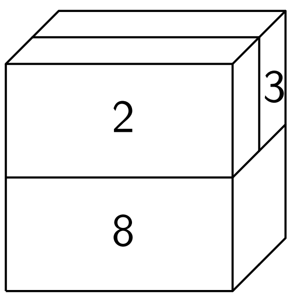

guillotine-cuts



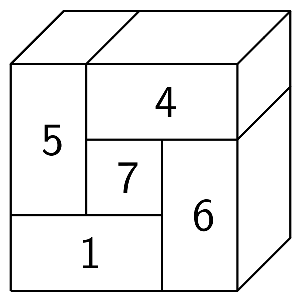

non guillotine-cuts

# Three-dimensional packing problems (brief outline)

- **Complexity**: obviously NP-hard in the strong sense.
- **Applications:**
  - *Loading*: containers, vehicles (trucks, freight cars), pallets;
  - *Packaging design*: boxes, cases;
  - *Cutting*: foam rubber (arm-chair production).
- **Variants and additional constraints:**
  - Guillotine cuts:




guillotine-cuts

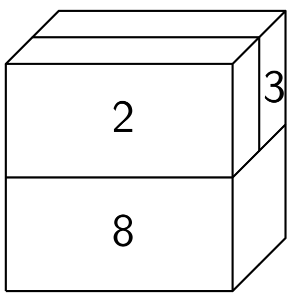



non guillotine-cuts

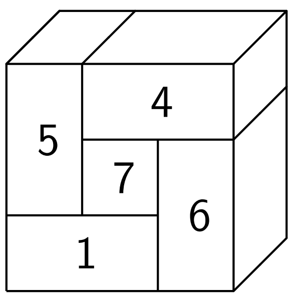
  - Boxes rotation;

## Three-dimensional packing problems (brief outline)

- **Complexity**: obviously NP-hard in the strong sense.
- **Applications:**
  - *Loading*: containers, vehicles (trucks, freight cars), pallets;
  - *Packaging design*: boxes, cases;
  - *Cutting*: foam rubber (arm-chair production).
- **Variants and additional constraints:**
  - Guillotine cuts:




guillotine-cuts

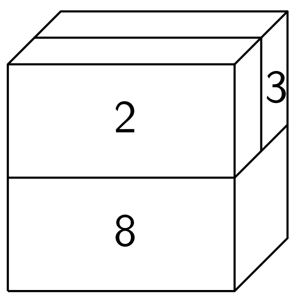



non guillotine-cuts

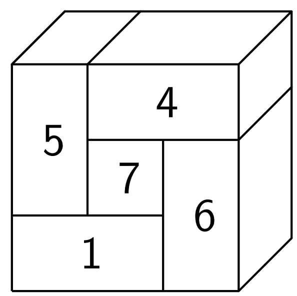
  - Boxes rotation;
  - Layers;

# Three-dimensional packing problems (brief outline)

- **Complexity**: obviously NP-hard in the strong sense.
- **Applications:**
  - *Loading*: containers, vehicles (trucks, freight cars), pallets;
  - *Packaging design*: boxes, cases;
  - *Cutting*: foam rubber (arm-chair production).
- **Variants and additional constraints:**
  - Guillotine cuts:




guillotine-cuts

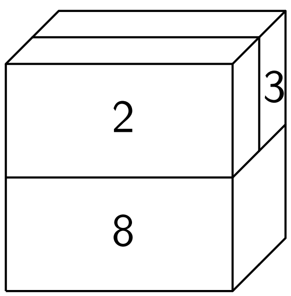



non guillotine-cuts

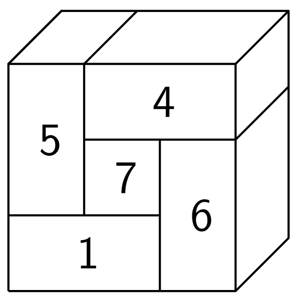
  - Boxes rotation;
  - Layers;
  - Limit on superposed weights;

## Three-dimensional packing problems (brief outline)

- **Complexity**: obviously NP-hard in the strong sense.
- **Applications:**
  - *Loading*: containers, vehicles (trucks, freight cars), pallets;
  - *Packaging design*: boxes, cases;
  - *Cutting*: foam rubber (arm-chair production).
- **Variants and additional constraints:**
  - Guillotine cuts:




guillotine-cuts




non guillotine-cuts

  - Boxes rotation;
  - Layers;
  - Limit on superposed weights;
  - Stability of the load . . .

# Essential Bibliography (Surveys and books)

# Essential Bibliography (Surveys and books)

## I. One-dimensional bin packing problem


– M. Delorme, M. Iori, S. Martello (2016). Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms, *European Journal of Operational Research*.

– E.G. Coffman, Jr, J. Csirik, G. Galambos, S. Martello, D. Vigo (2013). Bin packing approximation algorithms: Survey and classification
*Handbook of Combinatorial Optimization*, Springer.

– G. Wäscher, H. Haußner, and H. Schumann (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*.

# Essential Bibliography (Surveys and books)

## I. One-dimensional bin packing problem


– M. Delorme, M. Iori, S. Martello (2016). Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms, *European Journal of Operational Research*.

– E.G. Coffman, Jr, J. Csirik, G. Galambos, S. Martello, D. Vigo (2013). Bin packing approximation algorithms: Survey and classification
*Handbook of Combinatorial Optimization*, Springer.

– G. Wäscher, H. Haußner, and H. Schumann (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*.

– H. Kellerer, U. Pferschy, D Pisinger (2004). *Knapsack problems*, Springer, Berlin.

– S. Martello, P. Toth (1990). *Knapsack Problems: Algorithms and Computer Implementations* (Ch. 6), John Wiley & Sons, Chichester-New York. **Free download @ my home page**.

## II. Two-dimensional bin packing problems

– A. Lodi, S. Martello, M. Monaci, D. Vigo (2014). Two-dimensional bin packing problems. In *Paradigms of Combinatorial Optimization: Problems and New Approaches*, ISTE and John Wiley & Sons.

– A. Lodi, S. Martello, M. Monaci (2002). Two-dimensional packing problems: A survey. *European Journal of Operational Research*. ($>$ $900$ citations on Google Scholar)

## II. Two-dimensional bin packing problems


– A. Lodi, S. Martello, M. Monaci, D. Vigo (2014). Two-dimensional bin packing problems. In *Paradigms of Combinatorial Optimization: Problems and New Approaches*, ISTE and John Wiley & Sons.

– A. Lodi, S. Martello, M. Monaci (2002). Two-dimensional packing problems: A survey. *European Journal of Operational Research*. ($>$ 900 citations on Google Scholar)

– G. Scheithauer (2018). *Introduction to Cutting and Packing Optimization Problems*, Springer, Berlin.

## II. Two-dimensional bin packing problems


– A. Lodi, S. Martello, M. Monaci, D. Vigo (2014). Two-dimensional bin packing problems. In *Paradigms of Combinatorial Optimization: Problems and New Approaches*, ISTE and John Wiley & Sons.

– A. Lodi, S. Martello, M. Monaci (2002). Two-dimensional packing problems: A survey. *European Journal of Operational Research*. ($>$ 900 citations on Google Scholar)

– G. Scheithauer (2018). *Introduction to Cutting and Packing Optimization Problems*, Springer, Berlin.

– M. Iori, V. Loti de Lima, S. Martello, F.K. Miyazawa, M. Monaci (2019). *Two-dimensional Cutting and Packing: Problems and Solution Techniques* (in preparation).

## II. Two-dimensional bin packing problems


– A. Lodi, S. Martello, M. Monaci, D. Vigo (2014). Two-dimensional bin packing problems. In *Paradigms of Combinatorial Optimization: Problems and New Approaches*, ISTE and John Wiley & Sons.

– A. Lodi, S. Martello, M. Monaci (2002). Two-dimensional packing problems: A survey. *European Journal of Operational Research*. ($> 900$ citations on Google Scholar)

– G. Scheithauer (2018). *Introduction to Cutting and Packing Optimization Problems*, Springer, Berlin.

– M. Iori, V. Loti de Lima, S. Martello, F.K. Miyazawa, M. Monaci (2019). *Two-dimensional Cutting and Packing: Problems and Solution Techniques* (in preparation).


## III. Interactive visual solvers for packing problems


– M. Delorme, M. Iori, S. Martello (2018). BPPLIB: A library for bin packing and cutting stock problems. *Optimization Letters*.

– G. Costa, M. Delorme, M. Iori, E. Malaguti, S. Martello (2017). Training software for orthogonal packing problems. *Computers and Industrial Engineering*.

# It's been a long trip through packing

# It's been a long trip through packing

## It's been a long trip through packing

# Thank you for your attention