3D Bin Packing

Artificial Intelligence in Industry - Final Project
University of Bologna - Academic Year 2020/21

Leonardo Calbi, Lorenzo Cellini, Alessio Falai

3D-BPP

Problem definition

- We are given a set of n rectangular-shaped *items*, each characterized by width w_j, depth d_j and height h_j
- We are given a number of identical 3D containers (*bins*) having width W, depth D and height H
- The *three-dimensional bin-packing problem* (3D-BPP) consists in orthogonally packing all the items into the minimum number of bins

Assumptions

- Items may not be rotated
- We have unlimited bins $b_1, b_2, \ldots$ at our disposal
- All input data are positive integers satisfying $w_j \leq W, d_j \leq D, h_j \leq H$

Lower bounds

1. $L_0 = \left\lceil \frac{\sum_{j=1}^{n} v_j}{B} \right\rceil$, where $v_j = w_j \times d_h \times h_j$
   - Continuous lower bound: measures the overall *liquid volume*
   - Worst-case performance ratio of $\frac{1}{8}$
   - Time complexity: $O(n)$
2. $L_1 = \max\{L_1^{WH}, L_1^{WD}, L_1^{HD}\}$
   - Obtained by reduction to the *one-dimensional* case
   - Worst-case performance arbitrarily bad
   - Time complexity: $O(n)$
3. $L_2 = \max\{L_2^{WH}, L_2^{WD}, L_2^{HD}\}$
   - Explicitly takes into account the *three dimensions* of the items and dominates L_1
   - Worst-case performance ratio of $\frac{2}{3}$
   - Time complexity: $O(n^2)$

Dataset

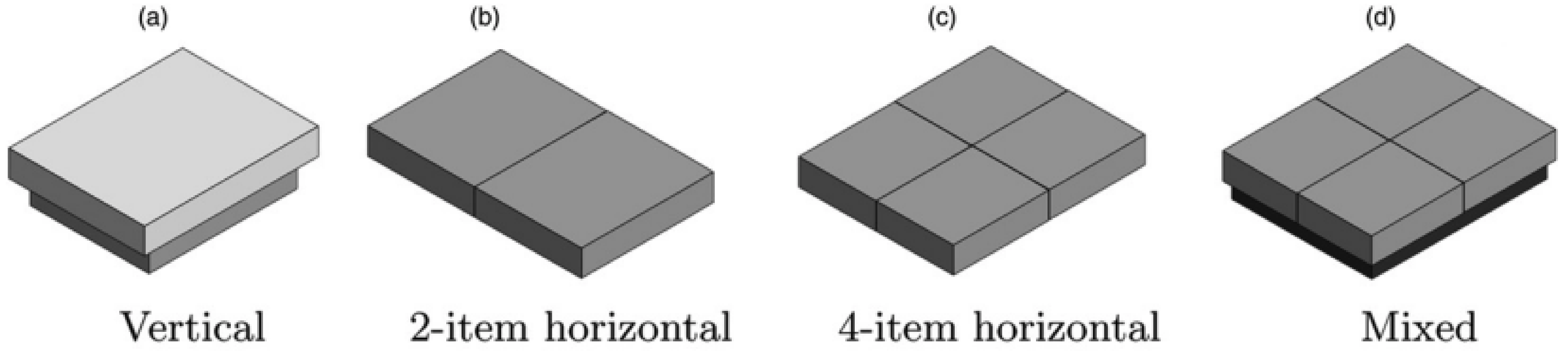
Distributions

Characteristic	Distribution	Parameters
Depth/width ratio R_{DW}	Normal	$(0.695, 0.118)$
Height/width ratio R_{HW}	Lognormal	$(-0.654, 0.453)$
Repetition F	Lognormal	$(0.544, 0.658)$
Volume V	Lognormal	$(2.568, 0.705)$
Weight L	Lognormal	$(2, 2)$

Reasoning

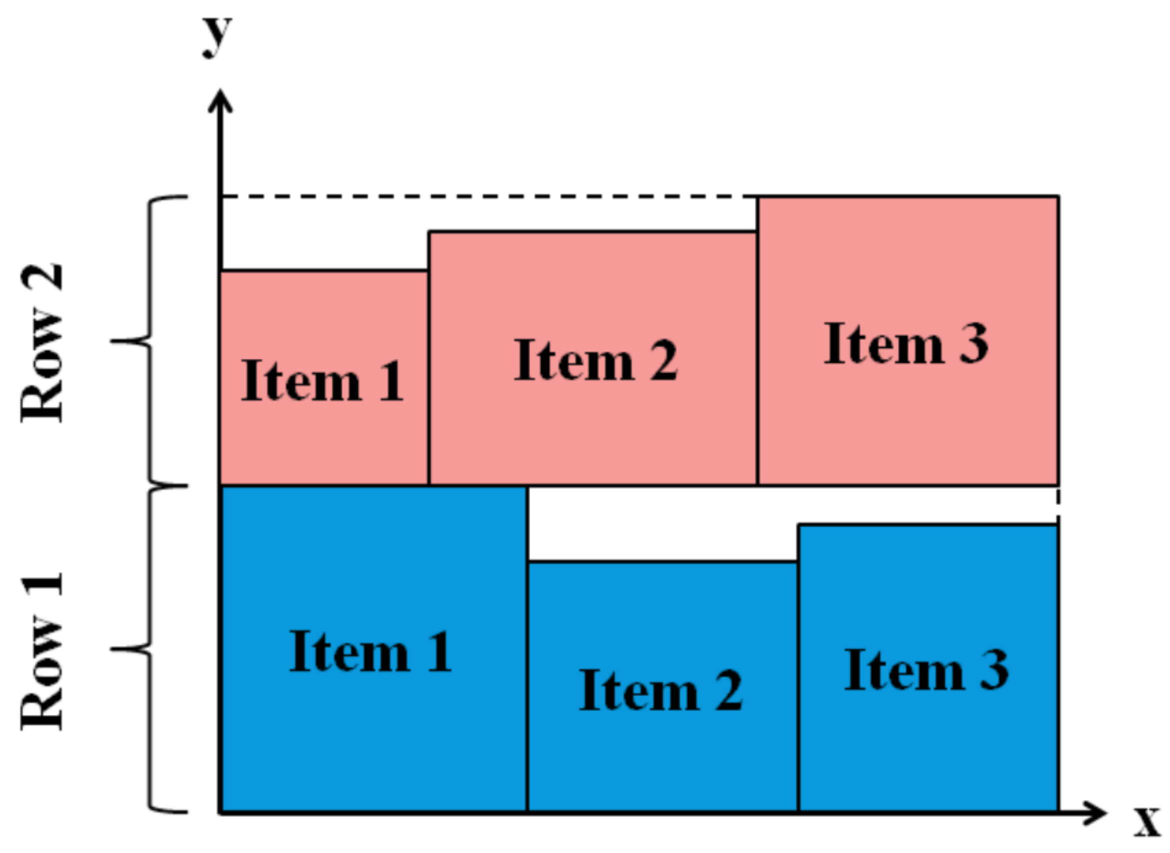
1. Volumes: $V \sim LN(\mu, \sigma^2), \mu = \frac{\sum_{j=1}^{N} \log v_j}{N}, \sigma^2 = \frac{\sum_{j=1}^{N} (\log v_j - \mu)^2}{N}, N = 166407, j \in \{C_1, \ldots C_5\}$
2. Widths: $W = (\frac{V}{R_{DW} \times R_{HW}})^{\frac{1}{3}}$
3. Depths: $D = W \times R_{DW}$
4. Heights: $H = W \times R_{HW}$

Superitems



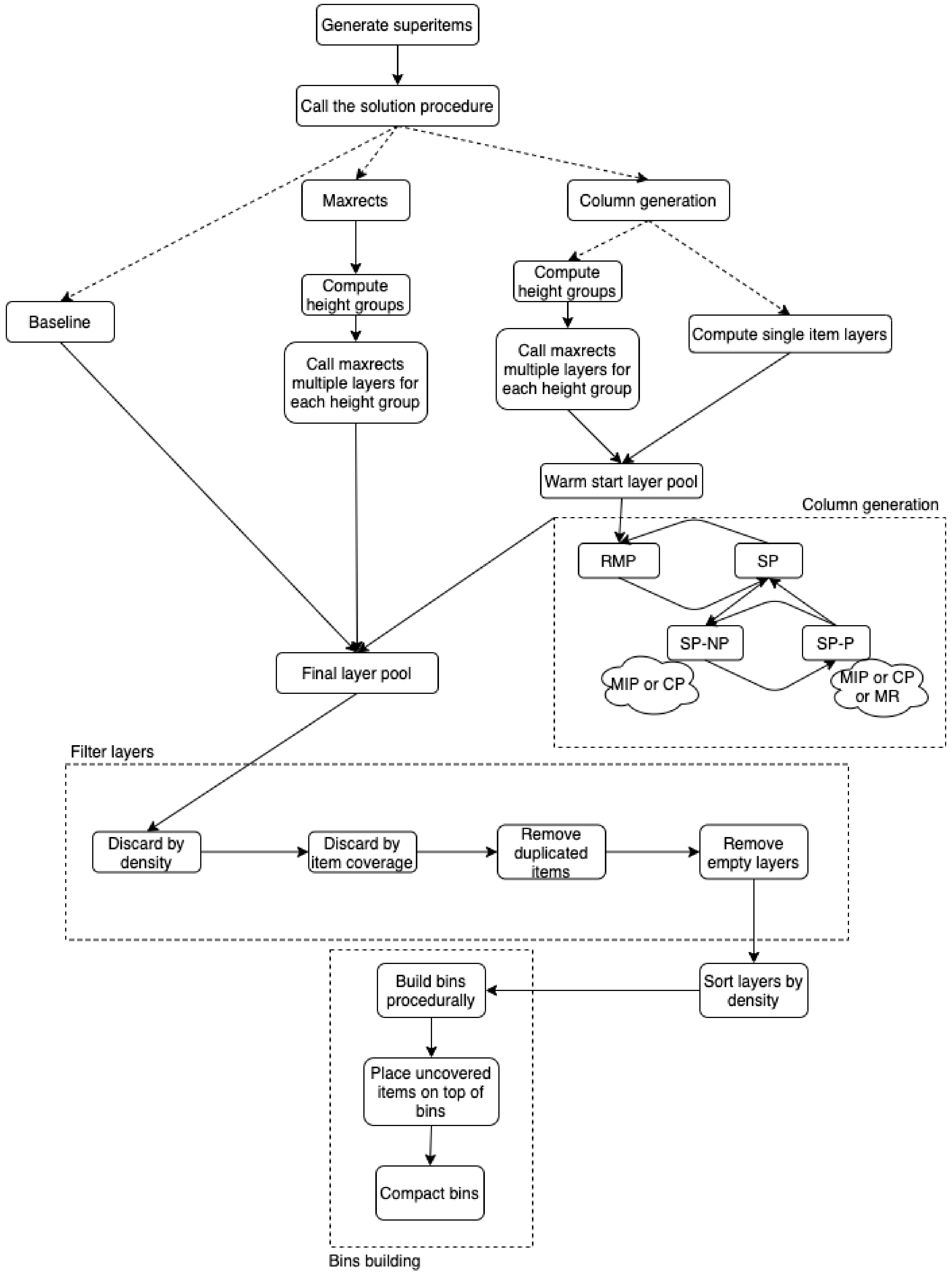
- A *superitem* is a collection of individual items that are compactly stacked together
- Building procedure
  1. *Single*: composed of a unique item
  2. *Horizontal*: composed of items having the exact same dimensions
     - Possibility to restrict their generation (*2D*, *2W*, *4* or none)
  3. *Vertical*: composed of items s.t. the ones on top have an area support of at least 70%
     - Maximum M stacked superitems (either single or horizontal)

Layers



- A *layer* is defined as a two-dimensional arrangement of items within the horizontal boundaries of a bin with no superitems stacked on top of each other
- Superitems are placed relative to layers and layers are placed relative to bins

Workflow



Generate superitems
Call the solution procedure
Baseline
Maxrects
Column generation
Compute height groups
Call maxrects multiple layers for each height group
Compute height groups
Call maxrects multiple layers for each height group
Compute single item layers
Warm start layer pool
Column generation
RMP
SP
SP-NP
SP-P
MIP or CP
MIP or CP or MR
Final layer pool
Filter layers
Discard by density
Discard by item coverage
Remove duplicated items
Remove empty layers
Sort layers by density
Build bins procedurally
Place uncovered items on top of bins
Compact bins
Bins building

Baseline

$$
\begin{aligned}
[SPPSI]:\ \min & \sum_{l\in\mathcal{L}} o^l \\
\text{s.t.} & \sum_{l\in\mathcal{L}}\sum_{s\in\mathcal{CI}} f_{si} z_{sl} = 1 && i\in\mathcal{I}, && (5)\\
& o^l \geq h_s z_{sl} && s\in\mathcal{CI}, l\in\mathcal{L}, && (6)\\
& \sum_{s\in\mathcal{CI}} w_s d_s z_{sl} \leq WD && l\in\mathcal{L}, && (7)\\
& x_{sj} + x_{js} + y_{sj} + y_{js} \geq z_{sl} + z_{jl} - 1 && j>s: s,j\in\mathcal{CI}, l\in\mathcal{L}, && (8)\\
& x_{sj} + x_{js} \leq 1 && j>s: s,j\in\mathcal{CI}, && (9)\\
& y_{sj} + y_{js} \leq 1 && j>s: s,j\in\mathcal{CI}, && (10)\\
& c_s^1 + w_s \leq c_j^1 + W(1 - x_{sj}) && s\neq j: s,j\in\mathcal{CI}, && (11)\\
& c_s^2 + d_s \leq c_j^2 + D(1 - y_{sj}) && s\neq j: s,j\in\mathcal{CI}, && (12)\\
& 0 \leq c_s^1 \leq W - w_s && s\in\mathcal{CI}, && (13)\\
& 0 \leq c_s^2 \leq D - d_s && s\in\mathcal{CI}, && (14)\\
& x_{sj}, y_{sj} \in \{0,1\} && s\neq j: s,j\in\mathcal{CI}, && (15)\\
& z_{sl} \in \{0,1\} && l\in\mathcal{L}, s\in\mathcal{CI}, && (16)\\
& o^l \geq 0 && l\in\mathcal{L}. && (17)
\end{aligned}
$$

Constraints

- (5): ensure that every item is included in a layer
- (6): define the height of layer l
- (7): redundant valid cuts that force the area of a layer to fit within the area of a bin
- (8): enforce at least one relative positioning relationship between each pair of items in a layer
- (9) and (10): ensure that there is at most one spatial relationship between items i and j along each of the width and depth dimensions
- (11) and (12): non-overlapping constraints
- (13) and (14): ensure that items are placed within the boundaries of the bin

`MAXRECTS`

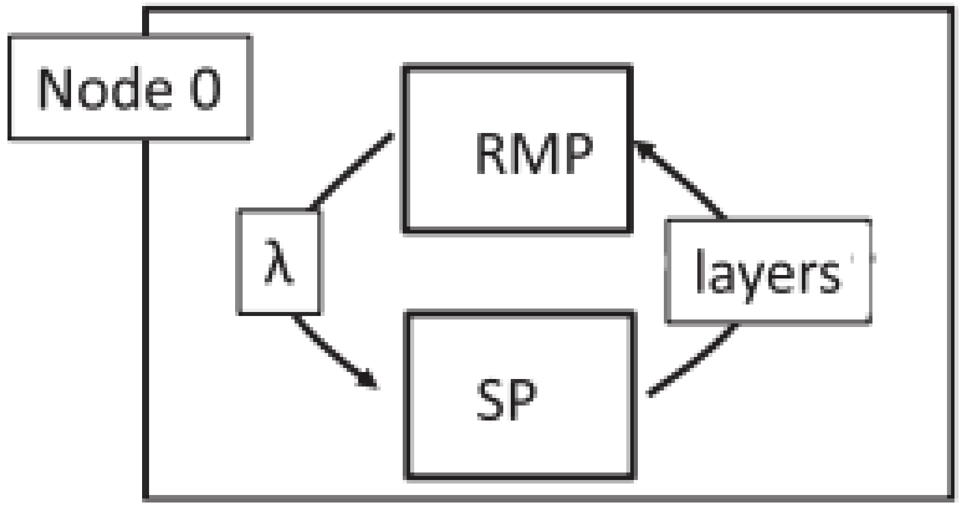
Algorithm 3: The Maximal Rectangles algorithm.

Initialize:
Set $\mathcal{F} = \{(W, H)\}$.
Pack:
foreach *Rectangle* $R = (w, h)$ *in the sequence* **do**
  Decide the free rectangle $F_i \in \mathcal{F}$ to pack the rectangle R into.
  If no such rectangle is found, restart with a new bin.
  Decide the orientation for the rectangle and place it at the bottom-left of F_i. Denote by B the bounding box of R in the bin after it has been positioned.
  Use the MAXRECTS split scheme to subdivide F_i into F' and F''.
  Set $\mathcal{F} \leftarrow \mathcal{F} \cup \{F', F''\} \setminus \{F_i\}$.
  foreach *Free Rectangle* $F \in \mathcal{F}$ **do**
    Compute $F \setminus B$ and subdivide the result into at most four new rectangles $G_1, \ldots, G_4$.
    Set $\mathcal{F} \leftarrow \mathcal{F} \cup \{G_1, \ldots, G_4\} \setminus \{F\}$.
  end
  foreach *Ordered pair of free rectangles* $F_i, F_j \in \mathcal{F}$ **do**
    if F_i *contains* F_j **then**
      Set $\mathcal{F} \leftarrow \mathcal{F} \setminus \{F_j\}$
    end
  end
end

Details

- `MAXRECTS` is a *procedural* algorithm for solving the 2D bin packing problem, based on an extension of the `GUILLOTINE` split rule
- *Height groups*: divide the whole pool of superitems into groups having heights within a given tolerance
- `MAXRECTS` is used to generate layers
- Run *multiple strategies* (Bottom-Left, Best Area Fit, Best Short Side Fit and Best Long Side Fit) and select the most dense layers

Column generation



- *Warm start*: single item layers vs `MAXRECTS`
- Each iteration builds only a *single layer* and adds it to the whole pool
- *Stopping criterion*: maximum iterations or convergence (non-negative reduced costs)
- *Optimality*: no branch-and-price scheme

`RMP`

$$
\begin{aligned}
[RMP]: \min \sum_{l \in \mathcal{L}'} & \alpha_l \bar{o}^l \\
\text{s.t.} \sum_{l \in \mathcal{L}'} \sum_{s \in \mathcal{C}\mathcal{I}} & f_{si} \bar{z}_{sl} \alpha_l \geq 1 \qquad i \in \mathcal{I}, \\
& \alpha_l \geq 0 \qquad l \in \mathcal{L}'.
\end{aligned} \tag{18}
$$

- `RMP` selects the best layers so far
- $\alpha_l \geq 0$ represents the linear relaxation of the integrality constraint $\alpha_l \in \{0, 1\}$
- λ are the dual variables corresponding to constraints (18)
- The master problem is solved using `BOP` (it only contains boolean variables), while the reduced one is solved with `GLOP` (linear program)

`SP`

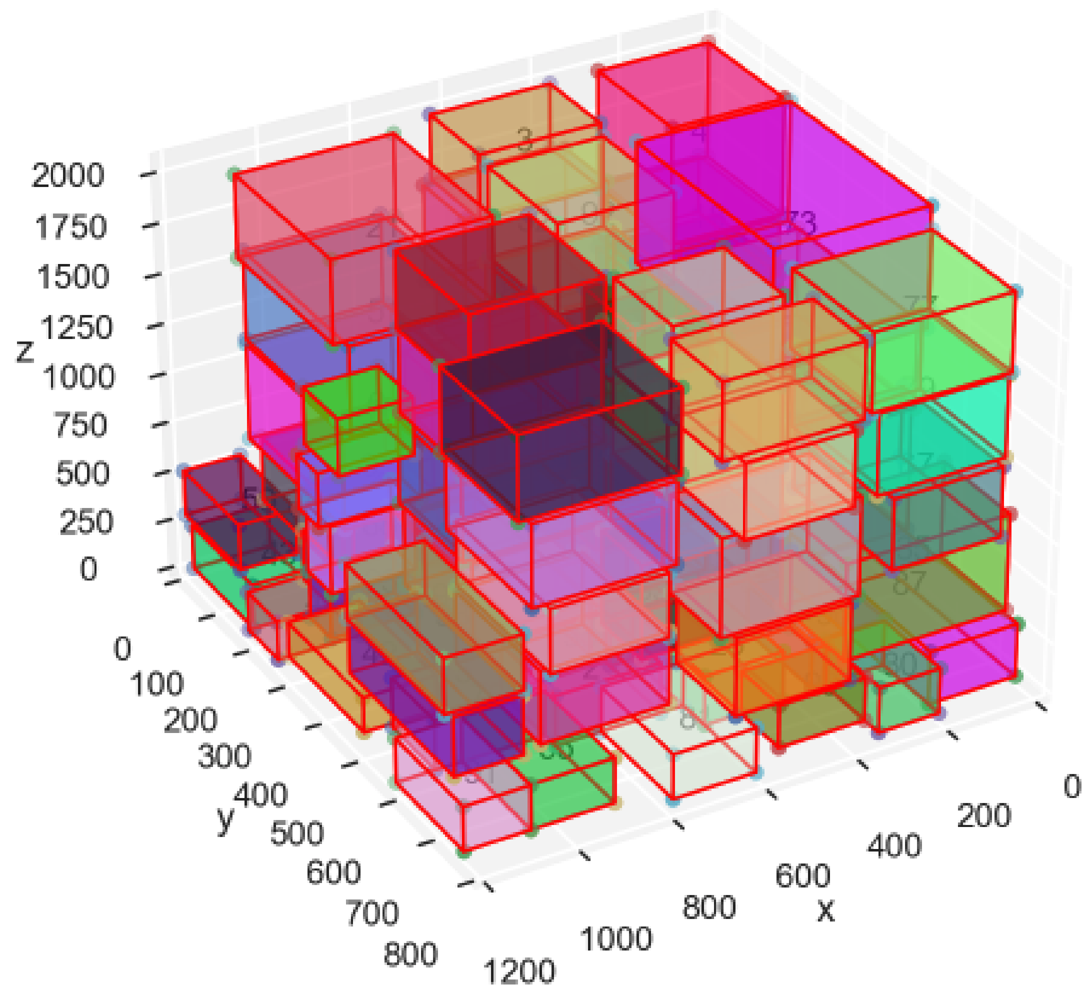
$$[SP] : \min \sum_{l \in \mathcal{L}} \left(o^l - \sum_{i \in \mathcal{I}} \sum_{s \in \mathcal{C}\mathcal{I}} \lambda_i f_{si} z_{sl} \right) \quad (19)$$
$$\text{s.t.} \ (6) - (17),$$

- `SP` selects items and positions them in a new layer
- $o^l - \sum_i \sum_s \lambda_i f_{si} z_{sl}$ is the reduced cost of a new layer l
- `SP` can be solved in the following ways
  - `MAXRECTS`: solve the whole pricing subproblem heuristically, using `MAXRECTS` to place superitems by biggest duals first
  - *Placement* and *no-placement* strategy
    1. *No-placement*: serves as an item selection mechanism, thus ignoring the expensive placement constraints (MIP or CP)
    2. *Placement*: checks whether there is a feasible placement of the selected items in a layer and places them if possible, otherwise iterates with the no-placement model (MIP or CP or `MAXRECTS`)

Layer filtering

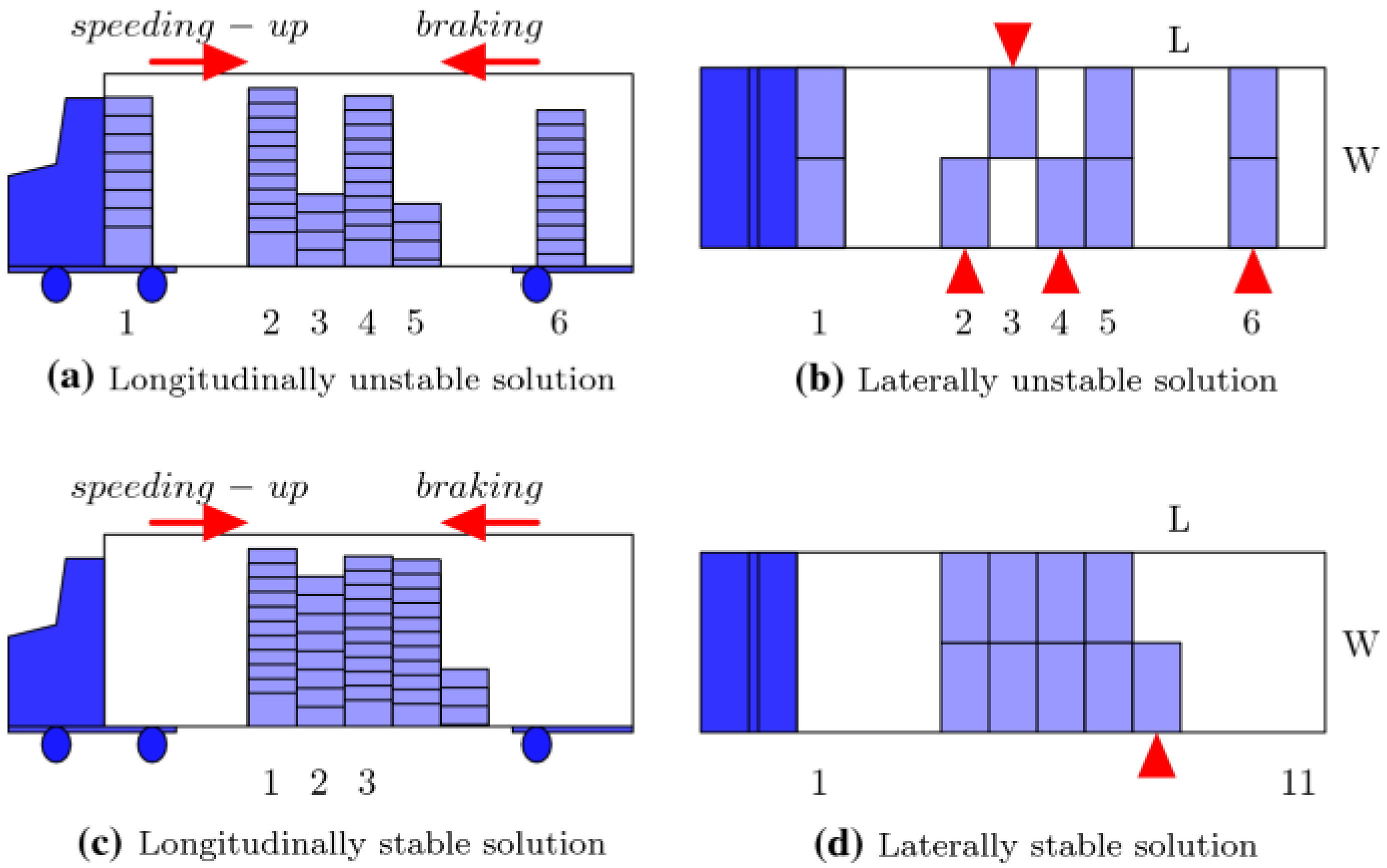
1. *Discard layers by densities* (given a minimum density d_m)
2. *Discard layers by item coverage*
   - If at least one item in layer l was already selected more times than M_a, discard l
   - If at least M_s items in layer l are already covered by previously selected layers, discard l
3. *Remove duplicated items*
   - Remove superitems in different layers containing the same item (remove the ones in less dense layers)
   - Remove superitems in the same layer containing the same item (remove the ones with less volume)
   - Re-arrange layers (using `` `MAXRECTS` ``) in which at least one superitem was removed (if $d_l > d_m$)
4. *Remove empty layers*
5. *Sort layers by densities*

Bin packing



1. *Uncovered items*: create new layers filled with items that were not covered in the previous procedures and add them to the layer pool
2. *Bins building procedure*: stack layers on top of each other until a bin is full and open new bins as needed
3. *Compact bins*: let items "fall" to the ground as much as possible, without allowing intersections

Future improvements



- Allow items to be *rotated* on the 3 axis
- Integrate the *branch-and-price* scheme into column generation to prove optimality
- Handle *weight* constraints and bin load capacity
- Improve item *support* through MP models (as described in the paper)
- Load bins inside *containers*

Demo

```
python3 -m streamlit run src/dashboard.py
jupyter notebook bpp.ipynb
```

END