POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Title

Tesi di Laurea Magistrale in
Xxxxxxx Engineering - Ingegneria Xxxxxxx

Author: **Name Surname**

Student ID: 000000
Advisor: Prof. Name Surname
Co-advisors: Name Surname, Name Surname
Academic Year: 20XX-XX

Abstract

Here goes the Abstract in English of your thesis followed by a list of keywords. The Abstract is a concise summary of the content of the thesis (single page of text) and a guide to the most important contributions included in your thesis. The Abstract is the very last thing you write. It should be a self-contained text and should be clear to someone who hasn't (yet) read the whole manuscript. The Abstract should contain the answers to the main scientific questions that have been addressed in your thesis. It needs to summarize the adopted motivations and the adopted methodological approach as well as the findings of your work and their relevance and impact. The Abstract is the part appearing in the record of your thesis inside POLITesi, the Digital Archive of PhD and Master Theses (Laurea Magistrale) of Politecnico di Milano. The Abstract will be followed by a list of four to six keywords. Keywords are a tool to help indexers and search engines to find relevant documents. To be relevant and effective, keywords must be chosen carefully. They should represent the content of your work and be specific to your field or sub-field. Keywords may be a single word or two to four words.

Keywords: here, the keywords, of your thesis

Abstract in lingua italiana

Qui va l'Abstract in lingua italiana della tesi seguito dalla lista di parole chiave.

Parole chiave: qui, vanno, le parole chiave, della tesi

Contents

1 | Introduction

Intro

Case study

Overview

Static stability

2 | Literature review

3 | Problem description and mathematical formulation

3.1. 3D Bin Packing Problem

3.2. Support

3.3. MILP Formulation

Conceptual model

A conceptual model of the problem we are trying to solve would be:

minimize	unused volume in used bins
subject to	all items assigned to one and only one bin
	all items within the bin dimensions
	no overlaps between items in the same bin
	all items with support

We can now provide the formal definition of the 3DBPP by formulating a mixed integer linear programming problem model.

Formal model

We'll now introduce a MILP model for the standard 3DBPP problem definition and then we'll expand it to address the stability constraint afterwards.

We start by defining the known sets and parameters of the problem.

Sets

$$I = \{1, \dots, n\}: \quad \text{set of items}$$
$$B = \{1, \dots, m\}: \quad \text{set of bins}$$

Parameters

$$\begin{aligned} W \times D \times H &\quad \text{width} \times \text{depth} \times \text{height of a bin} \\ V &\quad \text{bin volume} \\ w_i \times d_i \times h_i &\quad \text{width} \times \text{depth} \times \text{height of item } i \quad \forall i \in I \end{aligned} \tag{3.1}$$

Variables We can now introduce the following sets of integer variables

$$\begin{aligned} (x_i, y_i, z_i) &\quad \text{bottom front left corner of an item} && \forall i \in I \\ v_b &\quad \begin{cases} 1, \text{if bin } b \text{ is used} \\ 0, \text{otherwise} \end{cases} && \forall b \in B \\ u_{ib} &\quad \begin{cases} 1, \text{if item } i \text{ is placed in bin } b \\ 0, \text{otherwise} \end{cases} && \forall i \in I, \forall b \in B \\ x_{ij}^p &\quad \begin{cases} 1, \text{if } x_i + w_i \leq x_j \\ 0, \text{otherwise} \end{cases} && \forall i, j \in I \\ y_{ij}^p &\quad \begin{cases} 1, \text{if } y_i + d_i \leq y_j \\ 0, \text{otherwise} \end{cases} && \forall i, j \in I \\ z_{ij}^p &\quad \begin{cases} 1, \text{if } z_i + h_i \leq z_j \\ 0, \text{otherwise} \end{cases} && \forall i, j \in I \\ z_b^{\max} &\quad \text{maximum height of bin } b && \forall b \in B \end{aligned} \tag{3.2}$$

Given a coordinate system, each item i can be represented univocally in 3D space by section 3.3 as seen in figure 3.1

Objective function

$$\min \sum_{b \in B} (H v_b + z_b^{max}) \tag{3.3}$$

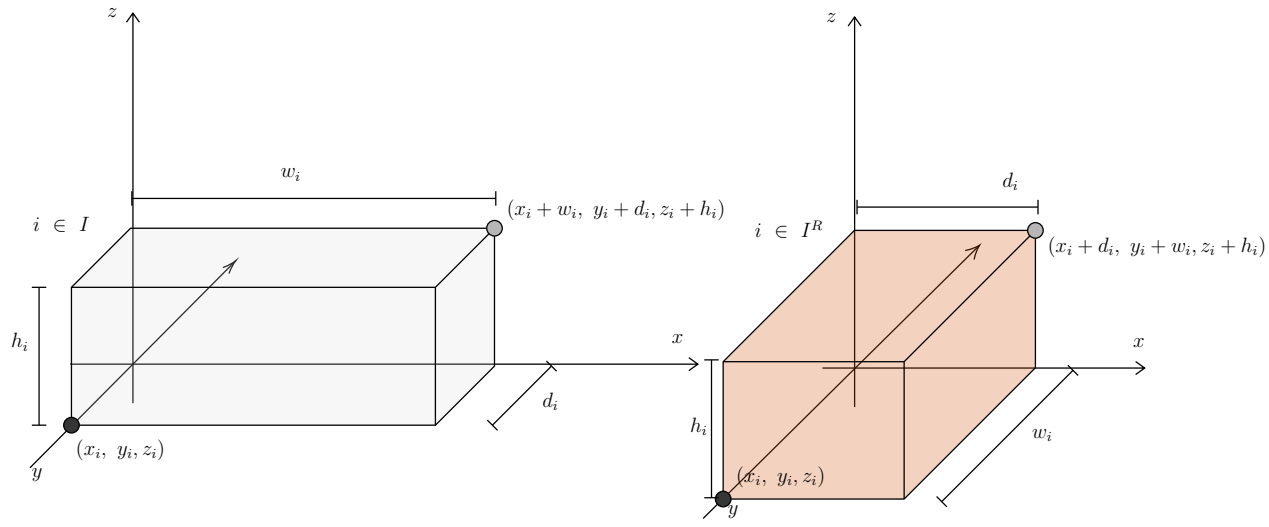

Figure 3.1: Coordinate system representation for a generic item i and it's rotated clone $i \in I^R$

Constraints

$$\sum_{b \in B} u_{ib} = 1 \qquad \forall i \in I \tag{3.4}$$

$$u_{ib} \leq v_b \qquad \forall i \in I, \forall b \in B \tag{3.5}$$

$$v_b \geq v_c \qquad \forall (b,c) \in B : b < c \tag{3.6}$$

$$x_i + w_i \leq W \qquad \forall i \in I \tag{3.7}$$

$$y_i + d_i \leq D \qquad \forall i \in I \tag{3.8}$$

$$z_i + h_i \leq H \qquad \forall i \in I \tag{3.9}$$

$$z_b^{max} \geq (z_i + h_i) - H(1 - u_{ib}) \qquad \forall i \in I, \forall b \in B \tag{3.10}$$

$$(x_i + w_i) - x_j \leq W(1 - x_{ij}^p) \qquad \forall i, j \in I \tag{3.11}$$

$$x_j - (x_i + w_i) + 1 \leq W x_{ij}^p \qquad \forall i, j \in I \tag{3.12}$$

$$(y_i + d_i) - y_j \leq D(1 - y_{ij}^p) \qquad \forall i, j \in I \tag{3.13}$$

$$y_j - (y_i + d_i) + 1 \leq D y_{ij}^p \qquad \forall i, j \in I \tag{3.14}$$

$$(z_i + h_i) - z_j \leq H(1 - z_{ij}^p) \qquad \forall i, j \in I \tag{3.15}$$

$$z_j - (z_i + h_i) + 1 \leq H z_{ij}^p \qquad \forall i, j \in I \tag{3.16}$$

$$x_{ij}^p + x_{ji}^p + y_{ij}^p + y_{ji}^p + z_{ij}^p + z_{ji}^p \geq u_{ib} + u_{jb} - 1 \qquad \forall i, j \in I, \forall b \in B \tag{3.17}$$

Othogonal rotations

Let us extend the definition of the bin packing problem without rotations with a new formulation which allows 90 degrees rotations of each item. Let $I = I^O \cup I^R$ be the new set of items where I^O is the set of original non-rotated items and I^R is the set of items rotated by 90 degrees.

We can now rewrite constraint 3.4 as 3.18 to force only one for the item between the original and rotated to be part of the solution.

$$\sum_{b \in B} u_{ib} + \sum_{b \in B} u_{jb} = 1 \quad \forall j = i^R \in I \tag{3.18}$$

Static stability constraints

We now extend the model introduced in section 3.3 to add constraints addressing static stability.

Additional Parameters

α_s : support area threshold

β_s : support height tolerance

δ : discretization unit

Additional Variables

$$s_{ij} \quad \begin{cases} 1, \text{if item } i \text{ can offer support to item } j \\ 0, \text{otherwise} \end{cases} \qquad \forall (i,j) \in I$$

$$z^c_{ij} \quad \begin{cases} 1, \text{if } 0 \leq z_j - (z_i + h_i) \leq \beta_s \\ 0, \text{otherwise} \end{cases} \qquad \forall (i,j) \in I$$

$$g_i \quad \begin{cases} 1, \text{if } z_i = 0 \\ 0, \text{otherwise} \end{cases} \qquad \forall (i,j) \in I$$

$$s_{ijb\delta_x\delta_y} \quad \begin{cases} 1, \text{if item } i \text{ receives support from item } j \\ \qquad \text{when both are placed in bin } b \\ \qquad \text{with distance } (\delta_x, \delta_y) \text{ between eachother} \\ 0, \text{otherwise} \end{cases} \qquad \forall (i,j) \in I$$

Computed Parameters

$$\gamma = \max_{\forall i \in I} \{w_i, d_i\} \quad : \quad \text{maximum possible dimension of an item}$$

$$\Delta = \left[-\left\lfloor \frac{\gamma}{\delta} \right\rfloor, \left\lfloor \frac{\gamma}{\delta} \right\rfloor \right] \quad : \quad \text{set of possible distances between overlapping items}$$

$$O : I \times B \times I \times B \times \Delta \times \Delta \to \mathbb{R}^+ \quad : \quad \text{discrete overlap function} \tag{3.19}$$

Additional Constraints

$$z_j - (z_i + h_i) \leq \beta_s + H(1 - z^c_{ij}) \qquad \forall (i,j) \in I : i \neq j \tag{3.20}$$

$$z_j - (z_i + h_i) \geq -\beta_s - H(1 - z^c_{ij}) \qquad \forall (i,j) \in I : i \neq j \tag{3.21}$$

$$s_{ij} \leq z^p_{ij} \qquad \forall (i,j) \in I \tag{3.22}$$

$$s_{ij} \leq z^c_{ij} \qquad \forall (i,j) \in I \tag{3.23}$$

$$s_{ij} \geq z^p_{ij} + z^c_{ij} - 2 \qquad \forall (i,j) \in I : i \neq j \tag{3.24}$$

$$\sum_{j \in I} s_{ij} \leq \sum_{b \in B} u_{ib} \qquad \forall i \in I \tag{3.25}$$

$$z_i \leq H(1 - g_i) \qquad \forall i \in I \tag{3.26}$$

$$\sum_{(\delta_x,\delta_y) \in \Delta, b \in B : O(i,b,j,b,\delta_x,\delta_y) \neq 0} s_{ijb\delta_x\delta_y} \leq s_{ij} \qquad \forall (i,j) \in I \tag{3.27}$$

We can then define the following set of constraints for $\forall (\delta_x, \delta_y) \in \Delta, \forall b \in B, \forall (i,j) \in I$:

$i \neq j \wedge O(i,b,j,b,\delta_x,\delta_y) \neq 0$

$$x_j - x_i \geq \gamma\delta_x - 2W(1 - s_{ijb\delta_x\delta_y}) \tag{3.28}$$

$$x_j - x_i \leq \gamma(\delta_x + 1) + 2W(1 - s_{ijb\delta_x\delta_y}) \tag{3.29}$$

$$y_j - y_i \geq \gamma\delta_y - 2D(1 - s_{ijb\delta_x\delta_y}) \tag{3.30}$$

$$y_j - y_i \leq \gamma(\delta_y + 1) + 2D(1 - s_{ijb\delta_x\delta_y}) \tag{3.31}$$

And finally we can introduce a feasibility constraint which ensures that every item that isn't on the ground is supported by some items in the same bin.

$$\sum_{(\delta_x,\delta_y)\in\Delta, b\in B, j\in I: i\neq j \wedge O(i,b,j,b,\delta_x,\delta_y)\neq 0} O(i,b,j,b,\delta_x,\delta_y) s_{ijb\delta_x\delta_y} \geq \alpha_s w_i d_i - w_i d_i g_i \quad \forall i \in I$$

4 | Solution algorithms

In this chapter, we describe a solution to the 3D bin packing problem with static stability. A solution candidate to the problem can be found by conducting a search over the tree of possible packings or states. In section 4.1 we describe what a state or packing is and its representation. Since an exhaustive search isn't feasible, a heuristic search is conducted by combining a beam search algorithm described in section 4.2 and a constructive heuristic described in section 4.3. The proposed algorithm takes in input an initial feasible state (as defined in section 4.1.2) usually represented by the empty state (4.3) and outputs the best scoring state based on an ordering function defined in section 4.2.1.

4.1. State

States or packings are partial solutions to the 3DBPP. Given the formal definition of the problem (3.3) a few new definitions are introduced to facilitate the algorithm's definition.

Definition 4.1 (Unpacked item). *An item $i \in I$ is unpacked* ***iff***

$$\sum_{b \in B} u_{ib} = 0$$

It is also assumed that variables identifying an item's position, and rotation are independent between states (changes to their values in state s won't affect state s').

A state s can then be defined as follows

- U: the set of unpacked items
- B: the set of bins
- $Q = (q_1, q_2, \ldots, q_b)$: the set of supporting structures for each bin $b \in B$
- p: the insertion pending on this state (described by def. 4.4)

Observation 4.1. *Given two states s and s' we can have that $|s.B| \neq |s'.B|$ since the number of bins is also a variable in the proposed heuristic*

We can also trivially define a function that determines if a state is a final state

Definition 4.2. *A state s is final if there are no more items to pack*

$$IsFinal(s) = \begin{cases} 1, & s.U = \emptyset \\ 0, & otherwise \end{cases} \tag{4.1}$$

Each bin b has additional data that is contained in a structure $q_b \in s.Q$ used to facilitate the execution of the algorithm.

Let us introduce the concept of packed items inside a bin:

Definition 4.3 (Packed item). *Given a state s and a bin $b \in s.B$, we say that item*

$$i \in I \text{ is packed in } b \textbf{ iff } \quad u_{ib} = 1$$

Given a bin $b \in s.B$ we can then define structure q_b as follows

- J: the set of items that are packed inside b
- Z: the set of planes inside b (section 4.3)
- T: the AABB Tree (section 4.1.1) representing the items inside b

Notice that two separate sets containing the items packed in b are present inside q_b but adding and accessing items in $q_b.J$ has a time complexity of $O(1)$ given an underlying implementation as HashSet while maintaining $q_b.T$ usually has a time complexity of $O(log(|q_b.J|))$.

The reason to include an AABB Tree inside this structure is further explained in sections 4.1.2 and 4.3.1

4.1.1. AABB Tree

In order to determine the feasibility of a given state, a way of checking for overlaps with items already placed is needed. Since our formulation of the problem only allows for 90 deg rotations over the z-axis. Every item in a solution, by the problem formulation (3.1), is contained inside a bounding box and this box is axis-aligned. An adequate structure to compute overlaps is then an Axis-Aligned Bounding Box Tree (AABB Tree) [1].

AABB Trees are bounding volume hierarchies typically used for fast collision detection and they usually offer a few operations:

- $AABBInsert(i)$: which allows inserting an axis-aligned box i in the tree
- $AABBOverlaps(i)$: which allows determining if an axis-aligned box i overlaps an element in the tree
- $AABBClosest(i, d)$: which, given an axis-aligned box i and a direction $d \in \{XP, XN, YP, YN, ZP, ZN\}$ along an axis, returns the closest element inside the tree following that direction starting from the box i

If the tree is properly balanced each operation on average has a time complexity of $O(log(n))$ where n is the number of elements in the tree. Maintaining an AABB Tree in the state allows us to do checks for feasibility during the construction of a solution (as detailed in 4.3.1) and feasibility checks on the final states to allow for error detection.

4.1.2. Feasibility

A state s is said to be feasible if the currently packed items for every bin $b \in s.B$ respects the constraints defined in the problem formulation (3.3)

Since the proposed heuristic is constructive it is more convenient to define the concept of feasibility relative to a change in the state.

Insertions Given a state s and $b \in s.B$, an insertion of items is a set of items that are placed in b and have their z_i within a tolerance of a certain z.

Definition 4.4 (Insertion). *Given a state s and a tolerance β_s we define an insertion or placement p a tuple (b, I) where b is a bin and I is a set of items that are going to be packed in b such that, $I \subseteq s.U \wedge \exists z(z \in \mathbb{Z} \wedge \forall i(i \in I \wedge |z_i - z| \leq \beta_s))$*

Observation 4.2. *Given s and $p = (b, \emptyset)$ where $b \notin s.B$, p is an insertion which will open bin b in s.*

Definition 4.5 (Next). *Let p be an insertion over a state s we can then define $s' = Next(s, p)$ as the "copy" of state s with $s'.p = p$. And p is then pending on s'.*

In this way, we can evaluate the changes to the score of a state based on its pending insertion without having to update all the structures for every evaluated state. This property will become apparent in section 4.2.

We can then define an algorithm that applies insertions to a given state s with pending insertions with the help of a function $OpenBin(b)$ which initializes a new structure q_b with every element at its empty value. The proposed algorithm is shown in 1.

Algorithm 1: Commit

input : s

output: s'

$(b, I) \leftarrow s.p$

$s' \leftarrow Clone(s)$ `//Memory clone of` s

if $b \in s'.B$ **then**

  $q_b \leftarrow (q_i \in s'.Q : i = b)$

  $q_b.J \leftarrow q_b.J \cup I$

  $s'.U \leftarrow s'.U \setminus I$

end

else Open a new bin

  $s'.B \leftarrow s'.B \cup b$

  $s'.Q \leftarrow s'.Q \cup OpenBin(b)$

end

$s'.p \leftarrow$ none

return s'

Insertion feasibility Describe insertion feasibility givine the sets defined

Algorithm 2: Is Insertion Feasible

input : $b, I, z, I_{support}, I_{upper}$

output: $isFeasible$

return *true*

State feasibility Describe how to compute the sets efficiently to use the insertion feasibility logic

Algorithm 3: Is State Feasible

input : s

output: $isFeasible$

return *true*

Proposition 4.1. *A state s' derived by committing a feasible insertion p to a feasible state s is feasible.*

Observation 4.3. *We can always define the empty state s_e where*

$$\begin{cases} s_e.U = I \\ s_e.Q = \emptyset \\ s_e.B = \emptyset \end{cases}$$

and it is always feasible

4.2. Beam Search

Beam Search (BS) is a heuristic tree search algorithm designed for systems with limited memory where expanding every possible node is unfeasible. The idea behind BS is to conduct an iterative truncated breadth-first search where, at each iteration, only a limited number of k nodes is expanded. After the expansion, every new node needs to be evaluated and sorted in order to prune the number of nodes down to the k best ones. The algorithm keeps exploring until no further node can be expanded.

To perform BS one must define the node structure, an expansion function to generate new nodes from existing ones, a ranking between nodes, and a function to determine if a node is final.

A node in the tree can be represented as the state in section 4.1 and eq. (4.1) can be used to determine if a state is final. We also know that a new state s' derived by s by applying a feasible insertion p can be computed as in definition 4.5. This state expansion procedure, with the exception of empty insertions, will generate new states in our tree which will add a positive number of bins or packed items to the solution so, eventually, it will generate a final state.

If the starting state for the search is feasible every new state generated will be feasible and if a final state is found it will be feasible (proposition 4.1). We also note that starting from state s the time complexity to compute feasible insertions can be lower than the complexity required to update the structures that will be used for further expansions (AABB Tree insertion and balancing, memory cloning, etc.) so we modified the standard BS algorithm to separate the expansion phase from the commit phase.

Given S^0 the set of initial states and k the number of best states to expand at each iteration, the described procedure is represented by algorithm 4. As observed in observation 4.3 it's possible to start the search from $S^0 = \{s_e\}$.

Algorithm 4: Beam search

input : S^0, k
output: s_{best}
$S^t \leftarrow S^0$
$S_{final} \leftarrow \emptyset$
repeat
  $S^{t+1} \leftarrow Expand(S^t)$ (algo. 5)
  $S_{final} \leftarrow S_{final} \cup \{s \in S^{t+1} : IsFinal(s)\}$ (def. 4.2)
  $S^{t+1} \leftarrow S^{t+1} \setminus S_{final}$
  $S^{t+1} \leftarrow Sort(S^{t+1})$ (sec. 4.2.1)
  $S^t \leftarrow \emptyset$
  $i \leftarrow 0$
  forall $s \in S^{t+1}$ **do**
    $S^t \leftarrow S^t \cup Commit(s)$ (algo. 1)
    $i \leftarrow i + 1$
    if $i > k$ **then**
      break
    end
  end
until $S^t \neq \emptyset$
$S_{final} \leftarrow Sort(S_{final})$
return *first element of* S_{final}

State Expansion An expansion of a state s can be seen as a new set of states S_{new} derived by a set of feasible insertions. In order to determine these insertions, an underlying heuristic is used (described in section 4.3).

The main idea in this phase of the algorithm is to find feasible insertions in all the bins for items that still need to be packed and that are of the same height. With this approach, the solutions given by the algorithm will start by trying to fill lower layers with items of the same height if possible and they'll become more heterogeneous in upper layers where the classes of height will start to mix up. The underlying heuristic will also use a scoring mechanism to select the best insertions for a given class of heights in order to avoid having too many states to sort.

Given a set of items I and a tolerance β_s we can introduce an algorithm to group them by their shap and produce a set G of tuples (h, I') where h is the hash summurizing the shape of the group and I' is the set of items grouped as in algo. **??**.

Once items are grouped by shape the best insertion for each class of items can be computed for each open bin. If no insertion is possible in any bin, then the only viable insertion is the bin opening insertion (observation 4.2). The described procedure is detailed in algo. 5.

Algorithm 5: Expand

input : S
output: S_{new}
forall $s \in S$ **do**
  $S_{new} \leftarrow \emptyset$
  $G \leftarrow GroupByHash(s.U)$ (algo. **??**)
  $placed \leftarrow false$
  forall $(h, I) \in G$ **do**
    forall $q_b \in s.Q$ **do**
      $P \leftarrow SPBestInsertion(q_b, I)$ (algo. 7)
      if $P \neq \emptyset$ **then**
        $placed \leftarrow true$
        forall $p \in P$ **do**
          $S_{new} \leftarrow S_{new} \cup Next(s, p)$ (def. 4.5)
        end
      end
    end
  end
  if $placed = false$ **then**
    `Open a new bin with index` $|s.B|$ `(oss. 4.2)`
    $S_{new} \leftarrow S_{new} \cup Next(s, (|s.B|, \emptyset))$
  end
end
return S_{new}

4.2.1. Scoring States

In order to sort states, a scoring function needs to be defined over them. Since the scoring of the states is what will influence the final solution the most, parameters that are directly related to minimizing the objective function are selected.

In the proposed solution to handle multiple objective functions, lexicographic ordering is used.

Algorithm 6: Group By Hash

```
input  : I
output: G
G ← ∅
forall i ∈ I do
    generate ← true
    forall (h, I') ∈ G do
        if h = hash(w_i, d_i, h_i) then
            generate ← false
            I' ← I' ∪ i
            break
        end
    end
    if generate = true then
        G ← G ∪ (hash(w_i, d_i, h_i), {i})
    end
end
return G
```

Definition 4.6. *Let $f_1(s), f_2(s), f_i(s), \ldots, f_n(s)$ be objective functions ordered by precedence based on index i, then*

$$s < s' \textit{ iff } \exists j \in \mathbb{Z} : \begin{cases} f_j(s) < f_j(s') \\ f_k(s) = f_k(s'), \quad \forall k \in \mathbb{Z} : 0 \leq k < j \end{cases}$$

Scoring metrics for each state s that we want to evaluate can then be computed in the *Next* algorithm by considering the contents of the pending insertions and updating each parameter differentially.

The defined ordering utilized is the following:

- $f_1(s) = -|s.B|$: we prefer states that opened fewer bins.
- $f_2(s) = \text{avgvol}(s)$: we prefer states that have packed more average volume between bins.
- $f_3(s) = \text{avgcageratio}(s)$: we prefer states that have better average cage ratio (def. **??**) between bins.

4.3. Support Planes

Support Planes (SP) is a heuristic detailed in the following section based on an underlying 2DBPP heuristic which is used to evaluate feasible insertions starting from a given state. Since the insertions must be feasible SP mantains an internal structure to facilitate the check for feasibility. The idea at the base of SP is to build a solution to the 3DBPP by filling 2D planes called *support planes*.

Each support plane can be characterized by the triple $S_z = (z, I_{support}, I_{upper})$ where

- z: the height of the plane
- $I_{support}$: the set of the items that can offer support to items placed on the plane
- I_{upper}: the set of items that will be obstacles to potential new items placed on the plane

Let *coords* be the set of possible coordinate changes which allow for the problem to evaluate placements starting from different corners of the bin.

Given a function $IsFeasible(i, bin, I_{support}, I_{upper}, aabb)$ which evaluates if a packing of item i in bin bin is feasible, and the function $ComparePacking(p, p')$ which defines a ranking over placements in the same plane, the SP algorithm can be written as algorithm 7.

Algorithm 7: SP Best Insertion

input : s_b, I

output: *placement*

$placement \leftarrow \emptyset$

forall $S_z \in planes$ **do**

  $I_p \leftarrow I \setminus \{i \in I : z + i.h > H_b\}$

  forall $change \in coords$ **do**

    $I'_{upper} \leftarrow CoordinateChange(change, I_{upper})$

    $I'_p \leftarrow CoordinateChange(change, I_p)$

    $P' \leftarrow SPPackPlane(W_b, D_b, I'_{upper}, I'_p)$ (Algorithm 8)

    $P \leftarrow CoordinateChange(change, P')$

    $P \leftarrow \{i \in P : IsFeasible(i, bin, I_{support}, I_{upper}, aabb)\}$

    if $ComparePacking(placement, P)$ **then**

      $placement \leftarrow P$

    end

  end

  if $placement \neq \emptyset$ **then**

    return *placement*

  end

end

return *placement*

To evaluate a packing on a plane a heuristic to solve the 2DBPP is used with the introduction of fixed placements which represent items on other planes that will be obstacles in the current one.

Given the dimensions of the 2D bin (W_b, D_b), the set of obstacles I_o and the set of items to pack I_p a new placement can be computed following algorithm 8

Algorithm 8: SP Pack Plane

input : W_b, D_b, I_o, I_p

output: P

$P \leftarrow \emptyset$

$2dPacking \leftarrow \emptyset$

foreach $i \in I_o$ **do**

    //Initialize the 2D bin packing instance with each obstable already placed

    $2DPlaceRect(2dPacking, i)$

end

repeat

    //Pack untill full

    $p \leftarrow 2DPackRect(2dPacking, W_b, D_b, i)$

    $P \leftarrow P \cup \{p\}$

until $p \neq \emptyset$

return P

Commit Extension We now describe an extension to *Commit* (algo. 1) to update the structures needed by SP.

When a plane is filled, new insertions become less likely to be feasible. To avoid evaluating planes where no insertion is possible a mechanism to prune dead planes can be introduced.

Since best insertions for a bin are always evaluated by considering lower planes first, if all the insertions in *Expand* (algo. 5) happened over a z_{min} then we can safely remove the opened planes with $z < z_{min}$ for that bin. Let us introduce a z_{min} variable carried over in q_b for each bin, which is updated during the *Expand* phase with the minimum z of all the insertions on bin b. Once the best states are computed and *Commit* is called we can then use its value to prune planes in each q_b. Other operations are also necessary in the *Commit* algorithm to allow SP to update its data structures accordingly to the insertion.

Given a state s and an insertion p where each packed item $i \in p.I$ in bin b has z_i within tolerance of z and the minimum height for the considered bin $q_b.z_{min}$. The algorithm which updates the structures for a given bin b is represented by algorithm 9. This new algorithm can be used as the last step of the *Commit* algorithm for each $b \in s'.B$.

Algorithm 9: SP Apply and Filter

input : $s, p, z, z_{min}, \beta_s$

output: s

$q_b \leftarrow (q_i \in s.Q : i = p.b)$

`//Filter bad planes`

$q_b.Z \leftarrow q_b.Z \setminus \{(z', I_{support}, I_{upper}) \in q_b.Z : z' < z_{min}\}$

`//Apply insertion`

forall $i \in p.I$ **do**

  $q_b.T \leftarrow InsertAABB(i, q_b.T)$ `//If balanced` $O(log(n))$

  $generate \leftarrow true$

  forall $(z', I_{support}, I_{upper}) \in q_b.Z$ **do**

    `//Based on the distance from the top of the item`

    $dz \leftarrow z' - (z_i + h_i)$

    if $0 \leq dz \leq \beta_s$ **then**

      $generate \leftarrow false$

      $I_{support} \leftarrow I_{support} \cup i$

    end

    else if $dz < 0$ **then**

      $I_{upper} \leftarrow I_{upper} \cup i$

    end

  end

  if $generate$ **then**

    $q_b.Z \leftarrow q_b.Z \cup (z_i + h_i, \{i\}, \emptyset)$

  end

end

return s

4.3.1. Scoring Insertions

5 | Computational experiments

Table 5.1: Summary of use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
Global	k=1	423.87	1.37	65.87	65.18	1.31	**70.70**
	k=5	1,597.54	1.34	69.19	185.22	1.29	**73.08**
	k=10	2,627.52	1.32	70.35	344.90	1.27	**73.56**
	k=20	5,373.79	1.34	70.78	620.95	1.27	**74.57**
	k=50	14,203.10	1.31	72.11	1,279.96	1.29	**74.61**
	k=100	26,934.21	1.31	73.23	2,340.37	1.26	**75.36**
	k=200	48,944.90	1.30	73.89	4,465.78	1.25	**76.39**
Class 0-19	k=1	187.25	1.15	64.10	54.95	1.05	**70.69**
	k=5	489.40	1.05	70.38	111.75	1.00	**75.36**
	k=10	861.30	1.05	71.94	182.20	1.00	**75.77**
	k=20	1,588.15	1.05	72.04	308.45	1.00	**76.60**
	k=50	3,896.40	1.05	73.07	690.80	1.00	**76.95**
	k=100	7,789.90	1.00	75.45	1,204.35	1.00	**78.46**
	k=200	15,817.20	1.05	74.99	2,192.75	1.00	**78.27**
Class 20-39	k=1	50.90	1.00	68.21	17.80	1.00	**73.66**
N = [50, 70]	k=5	138.40	1.00	71.92	39.20	1.00	**74.78**
	k=10	253.10	1.00	73.15	74.95	1.00	**75.28**
	k=20	483.85	1.00	73.86	124.30	1.00	**76.46**
	k=50	1,193.55	1.00	74.77	288.50	1.00	**77.02**
	k=100	2,358.50	1.00	75.08	535.30	1.00	**77.11**
	k=200	4,769.85	1.00	76.69	1,033.00	1.00	**78.64**
Class 40-59	k=1	292.35	1.30	65.62	60.55	1.25	**71.34**
N = [70, 120]	k=5	1,025.65	1.30	67.97	172.35	1.30	**72.53**
	k=10	1,910.60	1.30	68.46	304.25	1.25	**72.04**
	k=20	3,666.40	1.30	68.68	571.90	1.25	**74.01**
	k=50	7,649.95	1.25	71.32	1,152.40	1.25	**75.25**
	k=100	15,848.15	1.25	72.90	1,956.55	1.20	**75.67**
	k=200	32,420.40	1.25	73.29	3,472.50	1.20	**76.10**
Class 60-79	k=1	1,371.00	2.20	64.68	158.00	2.05	**69.11**
N = [120, 200]	k=5	5,751.95	2.15	66.66	531.80	1.95	**71.31**
	k=10	9,040.85	2.05	68.56	1,033.15	1.90	**72.69**
	k=20	19,116.60	2.15	67.81	1,881.70	1.90	**73.84**
	k=50	52,937.40	2.05	69.94	3,744.70	2.00	**71.25**
	k=100	98,271.55	2.10	70.04	7,010.65	1.90	**73.80**
	k=200	170,191.55	2.00	71.15	13,544.15	1.90	**75.01**
Class 80-99	k=1	217.85	1.20	66.74	34.60	1.20	**68.68**
	k=5	582.30	1.20	69.03	71.00	1.20	**71.41**
	k=10	1,071.75	1.20	69.65	129.95	1.20	**72.00**
	k=20	2,013.95	1.20	71.52	218.40	1.20	**71.97**
	k=50	5,338.20	1.20	71.44	523.40	1.20	**72.57**
	k=100	10,402.95	1.20	**72.68**	995.00	1.20	71.74
	k=200	21,525.50	1.20	73.30	2,086.50	1.15	**73.95**

6 | Conclusions and future developments

A final chapter containing the main conclusions of your research/study and possible future developments of your work have to be inserted in this chapter.

Bibliography

[1] G. v. d. Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of graphics tools*, 2(4):1–13, 1997.

A | Appendix A

Table A.1: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
1	k=1	403	1	69.54	218	1	69.82
	k=5	384	1	70.9	157	1	74.64
	k=10	502	1	71.47	151	1	74.64
	k=20	786	1	71.47	187	1	73.33
	k=50	1732	1	71.47	357	1	74.26
	k=100	3524	1	71.47	613	1	76.54
	k=200	6892	1	74.71	1020	1	74.64
2	k=1	266	2	48.2	67	1	77.19
	k=5	835	1	78.58	196	1	84.69
	k=10	1537	1	78.58	311	1	86.65
	k=20	2045	1	83.22	607	1	87.59
	k=50	5233	1	83.22	1706	1	87.84
	k=100	11422	1	83.22	3226	1	86.94
	k=200	22911	1	83.22	3860	1	85.87
3	k=1	169	1	73.36	62	1	65.48
	k=5	335	1	73.36	104	1	73.31
	k=10	532	1	73.36	141	1	72.73
	k=20	1003	1	73.36	245	1	74.86
	k=50	2621	1	73.46	457	1	74.51
	k=100	5209	1	73.46	897	1	75.02
	k=200	10781	1	74.2	1676	1	78.9
4	k=1	384	1	53.7	57	1	79.2
	k=5	1048	1	59.27	153	1	79.91
	k=10	1934	1	59.27	203	1	76.37
	k=20	3754	1	59.27	313	1	79.44
	k=50	9266	1	65.04	754	1	82.18
	k=100	18445	1	72.44	1467	1	82.18
	k=200	36636	1	72.44	2956	1	82.18
5	k=1	52	1	67.48	25	1	74.44
	k=5	192	1	73.22	75	1	76.16
	k=10	324	1	73.22	104	1	69.76
	k=20	641	1	73.22	144	1	69.18
	k=50	1613	1	73.22	255	1	68.65
	k=100	3466	1	73.22	518	1	68.65
	k=200	7149	1	73.22	1050	1	68.74

Table A.2: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
6	k=1	357	2	35.53	76	1	78.78
	k=5	939	1	65.88	196	1	79.74
	k=10	1638	1	74.23	419	1	82.46
	k=20	3257	1	73.74	624	1	80.73
	k=50	8533	1	73.74	1295	1	80.52
	k=100	16594	1	75.36	2019	1	78.53
	k=200	33658	1	75.36	3925	1	77.35
7	k=1	308	1	62.38	32	1	68.32
	k=5	774	1	62.38	98	1	71.36
	k=10	1052	1	69.06	148	1	81.03
	k=20	2003	1	69.06	299	1	82.35
	k=50	4828	1	71.32	697	1	79.09
	k=100	10009	1	71.32	1138	1	82.12
	k=200	19931	1	71.32	2289	1	82.12
8	k=1	50	1	74.2	36	1	79.27
	k=5	142	1	74.2	46	1	78.78
	k=10	240	1	76.51	66	1	83.85
	k=20	472	1	80.77	126	1	83.85
	k=50	1196	1	82.12	317	1	83.85
	k=100	2410	1	82.12	617	1	83.85
	k=200	4844	1	82.12	1212	1	83.85
9	k=1	188	1	67.28	41	1	69.6
	k=5	580	1	74.36	135	1	73.26
	k=10	989	1	74.36	319	1	81.8
	k=20	1795	1	75.21	364	1	77.87
	k=50	4573	1	78.8	1377	1	80.34
	k=100	8641	1	78.8	1557	1	76.19
	k=200	18028	1	78.8	3058	1	76.07
10	k=1	37	1	75.91	24	1	72.18
	k=5	229	1	76.34	65	1	74.73
	k=10	321	1	76.34	102	1	74.73
	k=20	645	1	76.34	186	1	74.73
	k=50	1641	1	76.34	413	1	80.24
	k=100	3177	1	76.34	685	1	79.76
	k=200	6562	1	76.34	1376	1	79.76

Table A.3: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
11	k=1	83	1	66.82	26	1	69.88
	k=5	227	1	66.82	88	1	73.04
	k=10	437	1	73.04	94	1	73.04
	k=20	901	1	73.04	182	1	73.73
	k=50	2163	1	74.9	374	1	72.55
	k=100	4271	1	75.09	656	1	70.51
	k=200	8965	1	75.09	1338	1	73.61
12	k=1	290	2	54.85	41	2	38.76
	k=5	905	2	49.36	149	1	79.67
	k=10	1746	2	49.36	153	1	77.74
	k=20	3048	2	40.42	304	1	76.43
	k=50	6768	2	40.91	526	1	77.53
	k=100	13867	1	73.25	1054	1	79.32
	k=200	29292	2	42.83	2136	1	79.32
13	k=1	161	1	53.61	23	1	68.76
	k=5	333	1	69.77	62	1	72.32
	k=10	585	1	69.77	120	1	73.12
	k=20	1140	1	70.64	156	1	64.05
	k=50	2821	1	70.64	420	1	65.88
	k=100	5610	1	70.64	833	1	73.76
	k=200	11427	1	75.46	1119	1	72.44
14	k=1	209	1	66.77	30	1	70.43
	k=5	512	1	66.77	71	1	69.51
	k=10	959	1	71.72	229	1	80.74
	k=20	1773	1	71.72	442	1	73.78
	k=50	4093	1	72	993	1	78.03
	k=100	8228	1	72	1915	1	77.62
	k=200	16602	1	75.58	2885	1	74.15
15	k=1	216	1	79.47	74	1	78.65
	k=5	710	1	79.47	161	1	66.78
	k=10	1415	1	80.62	245	1	70.17
	k=20	2661	1	80.62	365	1	77.37
	k=50	6673	1	80.62	865	1	80.52
	k=100	12879	1	80.62	1530	1	85.66
	k=200	25418	1	80.62	3552	1	85.66

Table A.4: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
16	k=1	114	1	68.8	125	1	80.6
	k=5	471	1	71.42	139	1	76.27
	k=10	808	1	71.42	265	1	76.27
	k=20	1529	1	72.13	473	1	78.77
	k=50	3901	1	72.92	1081	1	78.77
	k=100	7624	1	76.33	1654	1	79.27
	k=200	15602	1	76.33	3217	1	78.91
17	k=1	98	1	71.2	27	1	73.37
	k=5	263	1	77.41	61	1	71.09
	k=10	535	1	77.41	148	1	72.96
	k=20	1014	1	77.41	276	1	76
	k=50	2540	1	78.54	616	1	75.31
	k=100	4890	1	78.54	989	1	79.99
	k=200	10395	1	78.54	1790	1	79.92
18	k=1	108	1	60.55	36	1	69.39
	k=5	244	1	75.18	59	1	80.2
	k=10	434	1	75.18	127	1	78
	k=20	801	1	75.18	204	1	78
	k=50	1957	1	77.61	376	1	78.85
	k=100	3807	1	77.61	796	1	78.85
	k=200	7824	1	80	1575	1	77.54
19	k=1	113	1	67.04	52	1	66.58
	k=5	330	1	77.57	133	1	77.82
	k=10	623	1	77.57	172	1	59.29
	k=20	1289	1	77.57	435	1	77.44
	k=50	2977	1	77.57	589	1	74.24
	k=100	6064	1	77.57	1235	1	83.16
	k=200	12258	1	78.13	2461	1	83.16
20	k=1	139	1	65.36	27	1	63
	k=5	335	1	65.36	87	1	73.91
	k=10	615	1	66.36	127	1	70.1
	k=20	1206	1	66.36	237	1	72.51
	k=50	2799	1	66.92	348	1	65.81
	k=100	5661	1	69.53	688	1	71.21
	k=200	11169	1	75.42	1360	1	71.21

Table A.5: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	*B*	*CR*	*TT (ms)*	*B*	*CR*
21	k=1	33	1	67.44	18	1	77.23
	k=5	97	1	75.35	38	1	71.08
	k=10	166	1	75.35	60	1	72.69
	k=20	298	1	76.28	108	1	71.65
	k=50	741	1	76.28	173	1	69.42
	k=100	1399	1	76.28	349	1	69.42
	k=200	2870	1	76.28	692	1	71.4
22	k=1	29	1	72.11	9	1	72.94
	k=5	69	1	73.29	26	1	74.94
	k=10	141	1	74.21	50	1	74.14
	k=20	277	1	74.94	80	1	74.65
	k=50	706	1	74.94	184	1	75.91
	k=100	1385	1	78.81	354	1	75.91
	k=200	2802	1	78.81	716	1	75.91
23	k=1	34	1	63.84	13	1	81.33
	k=5	94	1	73.15	31	1	78.01
	k=10	173	1	75.99	57	1	77.42
	k=20	329	1	75.99	84	1	81.09
	k=50	808	1	76.28	211	1	83.06
	k=100	1594	1	76.28	381	1	83.06
	k=200	3164	1	79.14	743	1	84.51
24	k=1	26	1	73.86	8	1	73.12
	k=5	65	1	75.64	23	1	73.12
	k=10	110	1	79.19	40	1	78.76
	k=20	207	1	79.45	77	1	77
	k=50	511	1	81.14	173	1	70.35
	k=100	1076	1	81.14	349	1	70.35
	k=200	2148	1	81.14	682	1	79.63
25	k=1	82	1	70.7	37	1	71.53
	k=5	229	1	70.7	99	1	69.62
	k=10	431	1	70.7	181	1	69.62
	k=20	816	1	70.7	283	1	78.27
	k=50	2005	1	70.7	462	1	75.37
	k=100	4011	1	71.74	933	1	75.37
	k=200	8134	1	72.09	1759	1	75.37

Table A.6: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
26	k=1	35	1	71.69	16	1	73.01
	k=5	102	1	75.92	26	1	77.99
	k=10	194	1	75.92	49	1	76.17
	k=20	400	1	75.92	68	1	72.31
	k=50	1008	1	75.92	161	1	73.89
	k=100	2126	1	76.05	327	1	73.89
	k=200	4295	1	76.05	644	1	73.89
27	k=1	63	1	66.47	22	1	76.56
	k=5	164	1	70.9	38	1	77.73
	k=10	276	1	70.9	59	1	72.99
	k=20	522	1	70.9	105	1	77.81
	k=50	1291	1	71.76	249	1	77.81
	k=100	2563	1	71.76	483	1	77.81
	k=200	5463	1	77.81	947	1	77.81
28	k=1	55	1	68.54	17	1	77.78
	k=5	136	1	68.54	36	1	78.32
	k=10	236	1	70.2	59	1	79.3
	k=20	429	1	73.5	103	1	83.47
	k=50	1096	1	73.5	272	1	83.95
	k=100	2151	1	73.5	451	1	83.95
	k=200	4486	1	74.86	931	1	86.45
29	k=1	48	1	73.14	17	1	73.85
	k=5	144	1	75.49	41	1	74.03
	k=10	244	1	79.77	68	1	76.69
	k=20	462	1	79.77	132	1	77.94
	k=50	1104	1	81.33	459	1	80.83
	k=100	2230	1	81.33	706	1	84.8
	k=200	4511	1	84.64	1425	1	84.8
30	k=1	25	1	72.31	8	1	75.76
	k=5	111	1	72.31	18	1	75.63
	k=10	199	1	76.44	34	1	75.63
	k=20	349	1	76.44	71	1	75.76
	k=50	946	1	76.44	163	1	80.23
	k=100	1865	1	76.57	317	1	80.23
	k=200	3472	1	79.56	645	1	82.23

Table A.7: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	*B*	*CR*	*TT (ms)*	*B*	*CR*
31	k=1	37	1	59.65	14	1	74.05
	k=5	106	1	71.6	38	1	72.43
	k=10	191	1	72.36	61	1	72.43
	k=20	365	1	72.36	103	1	73.48
	k=50	918	1	75.08	247	1	80.34
	k=100	1745	1	75.08	406	1	77.23
	k=200	3442	1	77.46	798	1	78.02
32	k=1	61	1	65.02	24	1	69.44
	k=5	148	1	72.09	30	1	78.06
	k=10	253	1	73.71	45	1	72.09
	k=20	515	1	73.71	83	1	72.09
	k=50	1229	1	73.71	214	1	72.09
	k=100	2481	1	73.71	425	1	72.09
	k=200	5037	1	73.71	850	1	72.37
33	k=1	106	1	71.14	40	1	79.25
	k=5	239	1	75.74	64	1	78.21
	k=10	462	1	75.74	120	1	78.21
	k=20	854	1	78.73	192	1	78.21
	k=50	2130	1	78.73	412	1	83.68
	k=100	4239	1	78.73	837	1	83.68
	k=200	8519	1	80.04	1685	1	83.68
34	k=1	36	1	61.8	12	1	62.54
	k=5	95	1	64.99	50	1	69.19
	k=10	170	1	64.99	78	1	73.28
	k=20	321	1	67.87	150	1	73.28
	k=50	735	1	73.08	250	1	75.11
	k=100	1390	1	73.08	504	1	75.11
	k=200	2791	1	79.15	868	1	72.67
35	k=1	36	1	69.35	13	1	69.86
	k=5	106	1	69.35	37	1	71.2
	k=10	253	1	72.43	81	1	71.2
	k=20	480	1	74.53	85	1	73.06
	k=50	1277	1	74.58	188	1	71.65
	k=100	2378	1	74.58	373	1	71.65
	k=200	4929	1	74.58	755	1	71.65

Table A.8: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
36	k=1	121	1	66.81	28	1	75
	k=5	343	1	66.81	53	1	73.69
	k=10	618	1	66.81	180	1	78.01
	k=20	1136	1	67.11	291	1	82.68
	k=50	2664	1	71.94	711	1	76.98
	k=100	5253	1	73.12	1223	1	80.92
	k=200	10658	1	75.67	2579	1	81.79
37	k=1	44	1	73.94	13	1	68.3
	k=5	106	1	73.94	36	1	71.96
	k=10	184	1	73.94	92	1	74.65
	k=20	384	1	73.94	124	1	76.03
	k=50	909	1	73.94	270	1	76.34
	k=100	1883	1	73.94	461	1	68.16
	k=200	3836	1	73.94	907	1	79.57
38	k=1	39	1	70.69	17	1	74.76
	k=5	132	1	70.69	43	1	77.39
	k=10	259	1	70.69	83	1	77.39
	k=20	522	1	71.28	160	1	77.39
	k=50	1289	1	72.24	396	1	77.39
	k=100	2606	1	72.24	950	1	78.95
	k=200	5212	1	72.24	1208	1	79.17
39	k=1	24	1	62.25	7	1	76.42
	k=5	71	1	69.91	14	1	79.74
	k=10	121	1	71.76	23	1	79.74
	k=20	230	1	71.76	47	1	80.77
	k=50	559	1	71.76	110	1	80.77
	k=100	1065	1	71.76	216	1	80.77
	k=200	2097	1	74.15	444	1	80.77
40	k=1	84	1	63.4	23	1	70.51
	k=5	211	1	71.97	43	1	73.19
	k=10	381	1	71.97	79	1	75.09
	k=20	781	1	71.97	140	1	72.27
	k=50	1945	1	71.97	465	1	75.22
	k=100	3730	1	71.97	661	1	78.91
	k=200	7531	1	72.51	1382	1	81.13

Table A.9: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	*B*	*CR*	*TT (ms)*	*B*	*CR*
41	k=1	326	1	68.36	38	1	70.62
	k=5	697	1	74.25	104	2	52.93
	k=10	1062	1	75.63	122	1	74.21
	k=20	1948	1	77.68	262	1	78.73
	k=50	4790	1	77.68	668	1	74.45
	k=100	10170	1	79.15	1062	1	77.1
	k=200	20996	1	79.15	2013	1	78.91
42	k=1	149	1	70.28	20	1	67.43
	k=5	479	1	75.83	108	1	68.59
	k=10	849	1	75.83	214	1	73.76
	k=20	1623	1	75.83	374	1	75.83
	k=50	4004	1	75.83	907	1	81.2
	k=100	8093	1	75.83	1923	1	77.36
	k=200	16798	1	75.83	2991	1	77.3
43	k=1	174	2	64.74	96	2	65.13
	k=5	1628	2	71.89	370	2	64.66
	k=10	3028	2	71.89	515	2	58.88
	k=20	6518	2	70.91	1089	2	67.08
	k=50	9427	2	68.02	2335	2	80.84
	k=100	21206	2	71.12	3015	2	76.47
	k=200	67001	2	67.51	5429	2	77.53
44	k=1	367	1	61.69	61	1	71.18
	k=5	868	1	69.72	205	1	83
	k=10	1506	1	69.72	310	1	78.48
	k=20	3062	1	69.72	760	1	76.04
	k=50	7259	1	69.72	1567	1	80.62
	k=100	14367	1	69.72	3204	1	81.5
	k=200	29580	1	69.72	5594	1	80.33
45	k=1	604	2	46.66	65	1	75.62
	k=5	1502	2	44.05	359	1	83.12
	k=10	2890	2	43.78	529	1	83.12
	k=20	4877	2	38.4	810	1	78.72
	k=50	9384	1	76.94	2070	1	83.31
	k=100	22382	1	76.94	4241	1	86.62
	k=200	36854	1	79.15	6044	1	85.35

Table A.10: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
46	k=1	259	2	77.06	78	2	71.7
	k=5	2706	2	72.32	101	2	66.68
	k=10	4194	2	70.3	280	2	72.68
	k=20	8229	2	70.3	456	2	77.02
	k=50	14847	2	76.33	1201	2	73.15
	k=100	26144	2	76.15	1841	2	74.16
	k=200	45738	2	75.98	4028	2	75.2
47	k=1	395	1	70.04	73	1	72.15
	k=5	1082	1	70.04	188	1	75.28
	k=10	2024	1	70.04	439	1	72.72
	k=20	3741	1	70.04	737	1	78.82
	k=50	8428	1	70.04	1798	1	78.82
	k=100	17165	1	72.83	3143	1	78.82
	k=200	34689	1	72.83	6153	1	78.82
48	k=1	208	2	63.3	75	2	62.24
	k=5	1616	2	69.72	179	2	61.82
	k=10	3395	2	63.35	410	2	54.88
	k=20	6845	2	62.39	749	2	66.88
	k=50	7946	2	65.44	913	2	63.9
	k=100	20583	2	65.15	2056	2	64.02
	k=200	43066	2	70.5	4014	2	66.03
49	k=1	152	1	64.08	36	1	79
	k=5	617	1	69.12	79	1	76.7
	k=10	1140	1	72.53	128	1	76.11
	k=20	2039	1	72.53	252	1	79.88
	k=50	4622	1	74.03	603	1	79.88
	k=100	8767	1	74.03	1170	1	81.7
	k=200	16856	1	74.03	2384	1	81.7
50	k=1	421	1	63.37	63	1	70.85
	k=5	899	1	67.26	279	1	74.19
	k=10	1950	1	70.81	283	1	74.19
	k=20	3077	1	70.81	670	1	67.69
	k=50	6702	1	70.81	1476	1	74.15
	k=100	14720	1	72.07	1272	1	67.3
	k=200	31094	1	72.29	2618	1	67.3

Table A.11: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
51	k=1	199	1	70.66	62	1	71.17
	k=5	1189	1	71.25	157	1	70.5
	k=10	2375	1	71.25	297	1	69.81
	k=20	4786	1	71.25	1206	1	71.6
	k=50	11267	1	71.25	879	1	64.67
	k=100	20929	1	71.25	1761	1	64.67
	k=200	41807	1	73.87	3514	1	64.67
52	k=1	54	1	71.22	22	1	77.21
	k=5	170	1	74.69	51	1	80.62
	k=10	332	1	74.69	85	1	76.17
	k=20	589	1	76.01	169	1	78.28
	k=50	1589	1	76.01	424	1	80.14
	k=100	3017	1	77.1	517	1	81.3
	k=200	6764	1	77.1	1030	1	81.3
53	k=1	434	1	63.73	54	1	70.09
	k=5	1510	1	63.73	217	1	75.61
	k=10	2843	1	63.73	365	1	69.2
	k=20	5270	1	63.73	544	1	75.31
	k=50	12437	1	68.54	1198	1	77.54
	k=100	22733	1	69.71	1561	1	72.15
	k=200	46766	1	70.13	3089	1	71.75
54	k=1	889	2	71.9	75	2	71.76
	k=5	1611	2	74.52	186	2	77.16
	k=10	3338	2	74.52	452	2	70.11
	k=20	7081	2	72.86	771	2	77.55
	k=50	17401	2	72.55	1609	2	77.67
	k=100	36080	2	77.76	2773	2	71.73
	k=200	69866	2	71.95	5305	2	75.45
55	k=1	136	1	58.5	46	1	69.04
	k=5	455	1	70.35	89	1	71.56
	k=10	786	1	74.32	168	1	71.98
	k=20	1501	1	78.76	303	1	72.31
	k=50	3475	1	78.76	748	1	72.31
	k=100	7132	1	78.76	1376	1	72.53
	k=200	14863	1	79.02	1909	1	79.6

Table A.12: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
56	k=1	319	1	63.92	141	1	79.42
	k=5	828	1	71.46	236	1	77.95
	k=10	1476	1	74.87	420	1	80.12
	k=20	2840	1	74.87	550	1	76.63
	k=50	7310	1	74.87	1478	1	81.85
	k=100	15072	1	74.87	2233	1	75.5
	k=200	30574	1	78.73	3703	1	73.29
57	k=1	249	2	64.23	53	2	55.11
	k=5	743	2	47.8	137	2	58.35
	k=10	1343	2	45.86	295	2	48
	k=20	2089	2	42.54	398	2	48.64
	k=50	5008	2	44.26	1116	2	54.39
	k=100	13920	2	53.72	2074	1	82.3
	k=200	26749	2	51.17	2753	1	82.01
58	k=1	106	1	65	34	1	77.03
	k=5	809	1	65	135	1	76.3
	k=10	1525	1	67.19	195	1	76.3
	k=20	3043	1	67.99	328	1	74.82
	k=50	7037	1	67.99	539	1	76.89
	k=100	14984	1	71.01	1032	1	76.84
	k=200	29794	1	71.48	2064	1	77.89
59	k=1	64	1	69.12	54	1	78.11
	k=5	278	1	69.66	131	1	78.2
	k=10	658	1	69.66	250	1	77.18
	k=20	1229	1	75.21	440	1	76.43
	k=50	3024	1	75.21	643	1	77.6
	k=100	5983	1	75.21	1248	1	77.6
	k=200	11839	1	75.21	2491	1	77.6
60	k=1	342	1	64.54	65	1	71.84
	k=5	826	1	66.78	136	1	77.47
	k=10	1498	1	69.14	328	1	82.87
	k=20	2941	1	71.75	570	1	81.86
	k=50	7042	1	72.18	876	1	71.62
	k=100	13516	1	75.64	1629	1	73.71
	k=200	26714	1	80.19	2324	1	69.89

Table A.13: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
61	k=1	2470	3	61.14	194	2	75.03
	k=5	7566	3	61.46	458	2	79.58
	k=10	7705	2	71.35	882	2	74.15
	k=20	30277	3	57.73	2574	2	78.49
	k=50	41450	2	71.61	4022	2	79.87
	k=100	69808	3	58.95	6114	2	79.77
	k=200	197317	2	71.15	13254	2	77.35
62	k=1	2572	2	66.32	179	2	73.37
	k=5	10605	2	69.85	507	2	73.55
	k=10	17406	2	73.78	1380	2	73.49
	k=20	45291	2	66.46	2006	2	72.71
	k=50	44703	2	72.57	6606	2	77.29
	k=100	147556	2	69.57	7603	2	76.6
	k=200	103674	2	73.59	20459	2	76.31
63	k=1	1665	2	64.66	269	2	75.39
	k=5	11233	2	63.95	956	2	77.32
	k=10	13900	2	68.69	1656	2	77.37
	k=20	28438	2	72.53	2944	2	75.38
	k=50	99439	2	72.61	9805	2	75.62
	k=100	71230	2	72.6	15467	2	77.67
	k=200	137283	2	73.18	12330	2	78.89
64	k=1	2452	2	66.08	156	2	76.31
	k=5	5607	2	71.17	392	2	75.41
	k=10	4290	2	67.29	713	2	72.47
	k=20	7240	2	65.83	1005	2	76.43
	k=50	60548	2	69.7	2428	2	77.74
	k=100	70161	2	73.64	6411	2	79.81
	k=200	144912	2	73.64	10006	2	80.02
65	k=1	1391	2	61.23	126	2	62.06
	k=5	8206	2	61.29	482	2	62.92
	k=10	17117	2	64.36	1116	2	61.67
	k=20	15367	2	69.79	2358	2	65.05
	k=50	78715	2	63.81	3023	2	60.12
	k=100	146277	2	72.45	7925	2	61.3
	k=200	406884	2	63.72	15111	2	64.17

Table A.14: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
66	k=1	1904	3	58.8	175	2	81.55
	k=5	10568	3	67.91	485	2	79.54
	k=10	5507	3	65.99	1231	2	77.61
	k=20	11187	3	65.99	1804	2	78.44
	k=50	119767	3	56.88	3802	2	81.5
	k=100	256247	3	68.01	6740	2	80.58
	k=200	222528	2	77.34	13746	2	82.84
67	k=1	925	2	70.81	122	2	68.56
	k=5	5226	2	69.78	439	2	66.14
	k=10	10700	2	65.29	901	2	66.67
	k=20	10307	2	70.82	1418	2	75.99
	k=50	48933	2	69.91	2782	2	68.27
	k=100	72109	2	68.5	6219	2	70.04
	k=200	80496	2	72.21	13593	2	75.07
68	k=1	662	2	67.89	266	2	74.12
	k=5	2206	2	70.69	485	2	71.13
	k=10	3601	2	70.59	933	2	72.56
	k=20	6176	2	69.41	3152	2	75.02
	k=50	21427	2	72.94	3732	2	75.85
	k=100	46567	2	74.75	5835	2	73.27
	k=200	136385	2	72.67	13022	2	72.26
69	k=1	533	2	67.01	164	2	70.51
	k=5	2336	2	66.89	645	2	56.89
	k=10	3697	2	69.5	906	2	57.8
	k=20	34601	2	68.7	1629	2	63.27
	k=50	25976	2	68.5	2795	2	55.69
	k=100	45713	2	68.21	7779	2	57.47
	k=200	95060	2	69.61	11836	2	67.42
70	k=1	706	2	66.92	114	2	48.99
	k=5	3006	2	66.92	228	2	50.21
	k=10	9862	2	67.94	923	2	49.66
	k=20	18979	2	67.95	1259	2	47.25
	k=50	47210	2	67.95	3296	2	45.48
	k=100	93287	2	67.94	5671	2	50.66
	k=200	175227	2	69.96	10166	2	46.7

Table A.15: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	*B*	*CR*	*TT (ms)*	*B*	*CR*
71	k=1	1937	2	68.46	275	2	78.92
	k=5	6324	2	67.23	923	2	79.06
	k=10	12072	2	60.76	1845	2	77.33
	k=20	43571	2	69.7	2488	2	77.21
	k=50	107498	2	69.7	4843	2	79.95
	k=100	196533	2	74.01	9836	2	80.56
	k=200	165421	2	70.67	30888	2	77.79
72	k=1	3135	2	69.36	180	2	75.19
	k=5	12714	2	70.89	841	2	78.12
	k=10	13866	2	73.13	1489	2	75.76
	k=20	17588	2	74.99	2531	2	80.41
	k=50	63019	2	72.49	4930	2	82.07
	k=100	99912	2	73.73	8868	2	76.87
	k=200	202486	2	73.73	15071	2	80.41
73	k=1	543	2	62.87	106	2	59.24
	k=5	1435	2	61.18	434	2	49.78
	k=10	2961	2	61.18	912	1	81.44
	k=20	5741	2	61.18	1491	1	78.06
	k=50	14454	2	59.95	2519	2	61
	k=100	48160	2	53.75	3573	1	81.19
	k=200	55897	2	56.57	16018	1	81.52
74	k=1	1141	3	59.03	110	3	51.47
	k=5	2047	2	74.12	573	2	78.35
	k=10	11023	2	75.26	647	2	81.52
	k=20	21326	2	78.24	1739	2	78.18
	k=50	22942	2	73.59	3399	3	52.15
	k=100	99298	2	76.12	4867	2	80.29
	k=200	187444	2	76.64	5707	2	78.08
75	k=1	1626	3	67.53	146	3	56.95
	k=5	7600	3	63.87	548	2	79.31
	k=10	14729	3	63.87	1095	2	82.81
	k=20	22788	3	68.43	1881	2	79.28
	k=50	124694	3	76.24	3366	2	78.77
	k=100	205410	3	66.02	7725	2	80.9
	k=200	537763	3	65.18	24478	2	84.28

Table A.16: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
76	k=1	1376	2	63.96	177	2	74.41
	k=5	8926	2	74.31	519	2	72.4
	k=10	17086	2	74.31	751	2	78.52
	k=20	33521	2	74.31	1159	2	78.81
	k=50	60053	2	73.48	3003	2	84.23
	k=100	135048	2	73.09	7363	2	78.8
	k=200	231300	2	71.65	16926	2	80.54
77	k=1	1023	2	49.84	120	1	71.99
	k=5	4338	2	39.27	651	1	82.65
	k=10	5616	1	71.67	1016	1	76.75
	k=20	10976	2	40.22	2028	1	80.46
	k=50	29232	1	72.87	3568	1	75.93
	k=100	41278	1	77.2	8318	1	73.59
	k=200	88763	1	77.2	5868	1	80.87
78	k=1	265	2	69.04	49	2	63.24
	k=5	1194	2	71.41	262	2	67.07
	k=10	2336	2	65.69	571	2	68.79
	k=20	4516	2	69.82	902	2	68.92
	k=50	12694	2	69.79	2142	2	68.06
	k=100	37343	2	70.11	3265	2	71.6
	k=200	68154	2	69.01	5888	2	68.1
79	k=1	619	2	65.18	142	2	68.24
	k=5	2508	2	68	521	2	73.19
	k=10	4839	2	68	912	2	73.2
	k=20	9629	2	68	1748	2	73.2
	k=50	24561	2	68	2860	2	69.99
	k=100	59050	2	70.69	6388	2	68.51
	k=200	119607	2	68.91	8898	2	70.83
80	k=1	475	2	67.45	90	2	76.64
	k=5	1394	2	73	287	2	73.64
	k=10	2504	2	72.45	784	2	74.19
	k=20	4813	2	76.09	1518	2	74.19
	k=50	11433	2	76.26	1973	2	75.36
	k=100	24444	2	71.36	4246	2	76.51
	k=200	47230	2	76.39	7618	2	76.66

Table A.17: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	*B*	*CR*	*TT (ms)*	*B*	*CR*
81	k=1	8	1	71.74	9	1	73.14
	k=5	32	1	71.74	17	1	71.74
	k=10	49	1	73.14	27	1	71.74
	k=20	94	1	73.14	49	1	71.74
	k=50	226	1	73.14	99	1	71.74
	k=100	451	1	74.59	185	1	71.74
	k=200	916	1	74.59	345	1	74.59
82	k=1	30	1	62.99	67	1	73.56
	k=5	128	1	75.39	124	1	77.7
	k=10	242	1	75.39	244	1	77.7
	k=20	460	1	75.89	577	1	75.39
	k=50	1150	1	75.89	1658	1	76.65
	k=100	2312	1	75.89	3109	1	76.14
	k=200	4703	1	75.89	4769	1	76.65
83	k=1	6	1	63.86	4	1	59.07
	k=5	16	1	63.86	11	1	65.64
	k=10	31	1	63.86	19	1	65.64
	k=20	52	1	64.13	39	1	65.64
	k=50	125	1	65.64	87	1	65.64
	k=100	245	1	65.64	173	1	65.64
	k=200	489	1	65.64	347	1	65.64
84	k=1	3	1	66.1	2	1	73.29
	k=5	9	1	73.29	5	1	74.06
	k=10	17	1	73.29	8	1	74.06
	k=20	32	1	73.29	11	1	74.06
	k=50	75	1	73.29	10	1	74.06
	k=100	150	1	73.29	10	1	74.06
	k=200	307	1	73.29	10	1	74.06
85	k=1	24	1	69.56	21	1	66.42
	k=5	75	1	69.56	33	1	79.68
	k=10	140	1	69.56	54	1	79.68
	k=20	267	1	69.56	94	1	80.26
	k=50	670	1	69.56	208	1	80.26
	k=100	1311	1	69.56	378	1	80.26
	k=200	2645	1	69.56	798	1	80.26

Table A.18: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
86	k=1	34	1	82.3	17	1	75.61
	k=5	80	1	82.3	20	1	82.3
	k=10	133	1	82.3	28	1	82.3
	k=20	259	1	82.3	53	1	82.3
	k=50	604	1	82.3	104	1	82.3
	k=100	1174	1	82.3	180	1	82.3
	k=200	2400	1	82.3	359	1	82.3
87	k=1	250	1	68.79	111	1	71.46
	k=5	380	1	70.77	189	1	76.67
	k=10	617	1	70.77	138	1	70.1
	k=20	1072	1	72.52	274	1	75.49
	k=50	2504	1	76.27	771	1	78.3
	k=100	4903	1	76.27	1531	1	78.3
	k=200	9892	1	77.89	2544	1	77.07
88	k=1	109	1	63.76	53	1	77.67
	k=5	244	1	67.29	98	1	77.67
	k=10	375	1	69.53	145	1	76.85
	k=20	690	1	73.38	310	1	73.38
	k=50	1581	1	75.66	752	1	76.45
	k=100	3019	1	77.26	1447	1	76.45
	k=200	6146	1	77.26	2800	1	76.45
89	k=1	32	1	67.75	22	1	75.37
	k=5	210	1	75.72	24	1	66.46
	k=10	376	1	75.72	41	1	66.46
	k=20	601	1	75.72	76	1	75.37
	k=50	1500	1	75.72	176	1	75.37
	k=100	3224	1	75.72	321	1	75.37
	k=200	5858	1	76.95	619	1	76.59
90	k=1	12	1	80.24	10	1	80.24
	k=5	41	1	80.24	18	1	80.24
	k=10	76	1	80.24	20	1	80.24
	k=20	148	1	80.24	23	1	80.24
	k=50	372	1	80.24	20	1	80.24
	k=100	738	1	80.24	20	1	80.24
	k=200	1517	1	80.24	20	1	80.24

Table A.19: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
91	k=1	27	2	61.65	23	2	62.87
	k=5	167	2	60.72	51	2	67.36
	k=10	285	2	65.81	99	2	69.17
	k=20	596	2	67.21	189	2	70.48
	k=50	1239	2	64.03	422	2	73.88
	k=100	2370	2	71.91	631	2	64.22
	k=200	4631	2	71.91	1244	2	63.94
92	k=1	41	2	68.98	17	2	64.7
	k=5	102	2	68.5	46	2	60.91
	k=10	180	2	68.5	87	2	63.43
	k=20	343	2	68.5	200	2	64.45
	k=50	1081	2	68.5	481	2	64.45
	k=100	2487	2	75.09	848	2	62.44
	k=200	4615	2	74.62	1428	2	61.01
93	k=1	11	2	53.18	6	2	49.39
	k=5	29	2	60.58	13	2	63.43
	k=10	62	2	60.75	23	2	63.43
	k=20	114	2	60.97	44	2	62.4
	k=50	251	2	53.59	102	2	62.4
	k=100	480	2	53.59	207	1	70.47
	k=200	1141	2	53.59	408	1	70.85
94	k=1	7	2	60.41	6	2	61.33
	k=5	18	2	60.41	10	2	62.49
	k=10	32	2	60.41	17	2	64.19
	k=20	65	2	72.49	30	2	64.19
	k=50	166	2	72.49	72	2	64.19
	k=100	332	2	72.49	146	2	74.4
	k=200	675	2	75.2	247	2	71.17
95	k=1	2596	1	71.15	217	1	80.4
	k=5	5493	1	71.15	345	1	81.11
	k=10	10208	1	71.15	859	1	78.75
	k=20	19066	1	71.15	1031	1	76.3
	k=50	53469	1	71.15	2902	1	82.38
	k=100	101264	1	71.15	5778	1	82.38
	k=200	198827	1	71.15	11145	1	82.38

Table A.20: Use-case tests

Instance		Single Placement			Order by Hash		
		TT (ms)	***B***	***CR***	***TT (ms)***	***B***	***CR***
96	k=1	447	1	60.54	29	1	64.78
	k=5	1417	1	60.54	108	1	67.96
	k=10	2778	1	60.54	191	1	68.33
	k=20	5405	1	66.23	685	1	71.57
	k=50	13724	1	66.23	1133	1	68.28
	k=100	26596	1	67	2122	1	68.28
	k=200	55258	1	67	3725	1	69.27
97	k=1	625	1	56.36	40	1	70.47
	k=5	2977	1	61.41	228	1	69.38
	k=10	5436	1	61.41	459	1	68.66
	k=20	10287	1	62.02	458	1	66.34
	k=50	26357	1	63.42	988	1	58.96
	k=100	53737	1	65.76	1873	2	36.16
	k=200	123811	1	72.94	9152	1	74.29
98	k=1	78	1	79.05	23	1	64.42
	k=5	172	1	79.05	51	1	64.42
	k=10	288	1	81.04	94	1	77.16
	k=20	530	1	81.04	137	1	67.57
	k=50	1193	1	81.04	319	1	73.63
	k=100	2307	1	81.04	621	1	73.75
	k=200	4741	1	81.04	1182	1	77.16
99	k=1	8	1	71.74	9	1	73.14
	k=5	26	1	71.74	17	1	71.74
	k=10	55	1	73.14	27	1	71.74
	k=20	97	1	73.14	53	1	71.74
	k=50	234	1	73.14	93	1	71.74
	k=100	464	1	74.59	186	1	71.74
	k=200	921	1	74.59	338	1	74.59
100	k=1	9	1	54.55	6	1	56.35
	k=5	30	1	56.35	12	1	67.25
	k=10	55	1	56.35	19	1	70.44
	k=20	101	1	67.45	35	1	70.44
	k=50	243	1	67.45	71	1	70.44
	k=100	495	1	70.22	134	1	70.44
	k=200	1017	1	70.44	250	1	70.44

B | Appendix B

It may be necessary to include another appendix to better organize the presentation of supplementary material.

List of Figures

List of Tables

List of Symbols

Variable	Description	SI unit
$\boldsymbol{u}$	solid displacement	m
$\boldsymbol{u}_f$	fluid displacement	m

Acknowledgements

Here you might want to acknowledge someone.