POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

Three-dimensional bin packing with vertical support

Tesi di Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: **Jacopo Libè**

Student ID: 952914
Advisor: Prof. Ola Jabali
Co-advisors: Davide Croci
Academic Year: 2021-2022

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

The problem addressed by our contribution is the 3D-SBSBPP (3D Single Bin-Size Bin Packing Problem) which is strongly NP-hard and is the generalization of the one-dimensional bin packing problem [11].

Case study

Overview

2 | Literature review

Static stability

Static stability or vertical support received most of its contributions from the fields of Pallet Loading Problems (PLP) and Cargo Loading Problems (CLP). In [4] a second order cone programming formulation was provided as a solution to a spacing problem between layers of a pallet. The pubblication further described the concept of minimizing the area of overlap between items of different layers to increase spacing between items of the same layer and increase the amount of potential support for higher layers. In [12] a formulation of the CLP was given with practical constraints like weight distrubtiom, non-regular container shapes and vertical stability through vertex support.

3 | Problem description and mathematical formulation

In this thesis, we address the 3D single bin-size bin packing problem (3D-SBSBPP) with the addition of a few practical constraints. Starting from a set of items of different sizes the goal is to arrange them in the least amount of bins of a given fixed size without any overlap between each other. In addition to the standard formulation of the problem, three practical constraints need to be taken into account:

- each item inside a bin should have static stability, meaning that every item should be supported either by the ground or by other items in the same bin,
- the cage ratio of each used bin should be maximized,
- each item can be rotated orthogonally along its vertical axis.

Given a certain placement of items inside a bin of base $W \times D$ with the top of the highest item being at $z_{\max}$ and the sum of the volume of each item being V, the bin's cage ratio is defined as eq. (3.1).

$$\mathrm{CR} = \frac{V}{W \cdot D \cdot z_{\max}} \tag{3.1}$$

A high cage ratio means that even if a bin isn't fully occupied, it could be used as a base for other structures. This property is desirable in some industrial settings. It is also noted that in a single bin configuration, maximizing cage ratio is equivalent to minimizing $z_{\max}$. Finally, a visual representation of the cage ratio metric is provided in fig. 3.1.

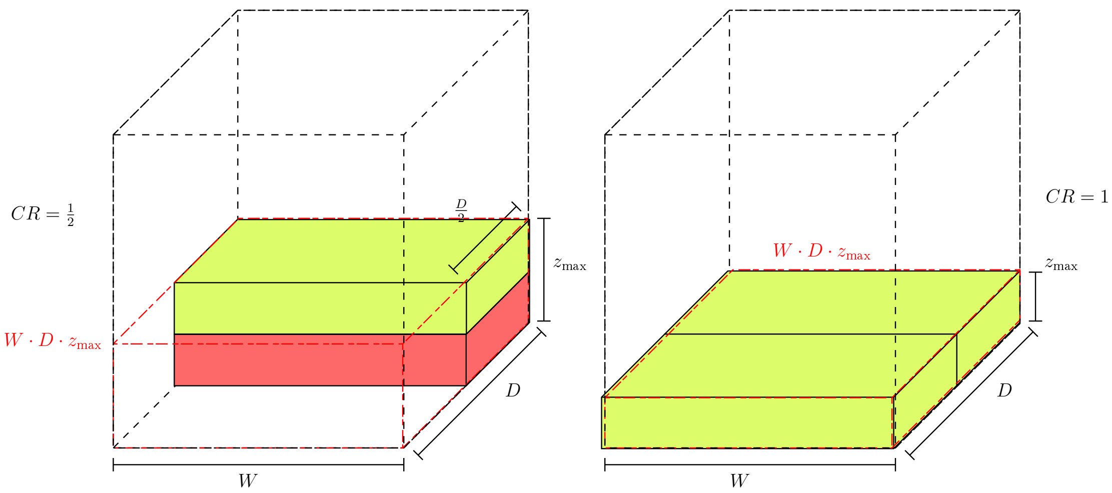


Figure 3.1: Cage ratio of two different bin configurations

Our notion of vertical support stems from rules imposed by the industry and from the literature on Pallet Loading Problems and Container Loading Problems. Vertical stability is usually ensured between horizontal or vertical slices of items as a constraint on the minimum amount of area which rests on other items (as for ex. [4, 9, 12]). Given a support area threshold which is usually equal to $\alpha_s = 0.7$ and a tolerance, which in our case is equal to $\beta_s = 1\text{cm}$, we can define an item as supported if

1. the sum of the overlap area over the XY-plane with every other item on which it is resting is greater than α_s times its base area. (area support)
2. the number of its corners resting on another item is greater than 3, and condition 1 holds with a lower threshold $\alpha'_s < \alpha_s$. (vertex support)

A visual representation of the condition of support is illustrated in fig. 3.2.

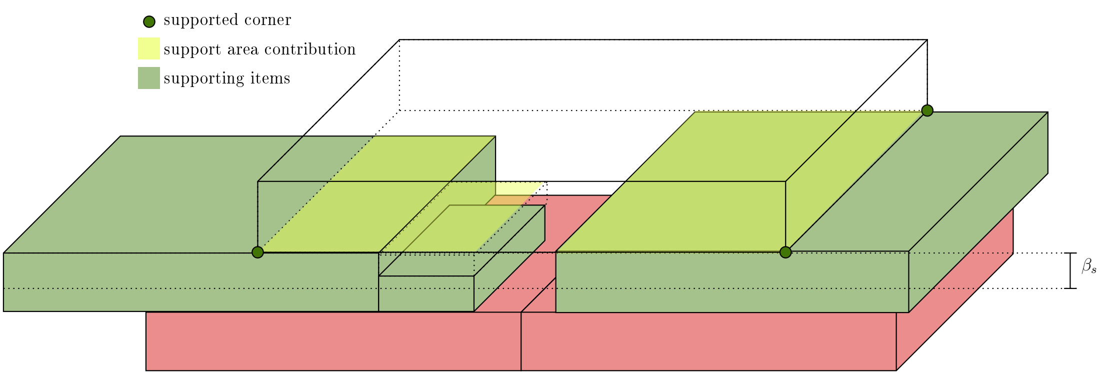


Figure 3.2: Representation of an item with vertical support given $\alpha_s = 0.5, \beta_s$

Given the definitions of our practical constraints, a conceptual formulation of our model would be

$$
\begin{array}{rc}
\textbf{minimize} & \text{number of used bins} \\
 & \text{the unused volume of each bin under } z_{\max} \\
\textbf{subject to} & \text{all items assigned to one and only one bin} \\
 & \text{all items within the bin dimensions} \\
 & \text{no overlaps between items in the same bin} \\
 & \text{all items with vertical support}
\end{array}
$$

In section 3.1 a mixed-integer linear programming model for the 3D-SBSBPP is presented and it's later extended to include orthogonal rotations in section 3.1.1 and vertical support constraints limited to condition 1 (area support) in section 3.1.2. Cage ratio isn't directly included in the proposed MILP formulation since the evaluation of the heuristic with the model was done in a single bin configuration where minimizing the maximum height of the bin is equivalent to minimizing the cage ratio.

3.1. 3D single bin-size bin packing problem

Let $I = \{1, \ldots, n\}$ be the set of items that need to be packed, $B = \{1, \ldots, m\}$ the set of bins to evaluate of fixed dimensions $W \times D \times H$. Each item $i \in I$ is characterized by a given width, depth, and height (w_i, d_i, h_i). Let us introduce three continuous variables that identify the position of an item's bottom front left corner (x_i, y_i, h_i) as seen in fig. 3.3. We can now introduce a set of integer variables v_b which will be 1 if bin $b \in B$ will be used in the solution and 0 otherwise. A set of integer variables u_{ib} which will be 1 if item $i \in I$ will be placed in bin $b \in B$ and 0 otherwise. To check for overlaps, we introduce three sets of integer variables for each axis of possible overlap to determine if there is a clear order of precedence on at least one axis. This formulation is also usually used in scheduling problems. One of the introduced sets is the set x^p_{ij}, which will take the value of 1 if item $i \in I$ precedes item $j \in I$ over axis x and 0 otherwise. This condition is verified if $x_i + w_i \leq x_j$. The other two sets are defined in a similar way over the remaining axis y^p_{ij} and z^p_{ij}. An additional set of continuous variables $z^{\max}_b$ is introduced which will assume the value of the maximum $x_i + h_i$ of the items $i \in I$ placed in bin $b \in B$.

The 3D-SBSBPP can then be formulated as a mixed-integer linear programming problem:

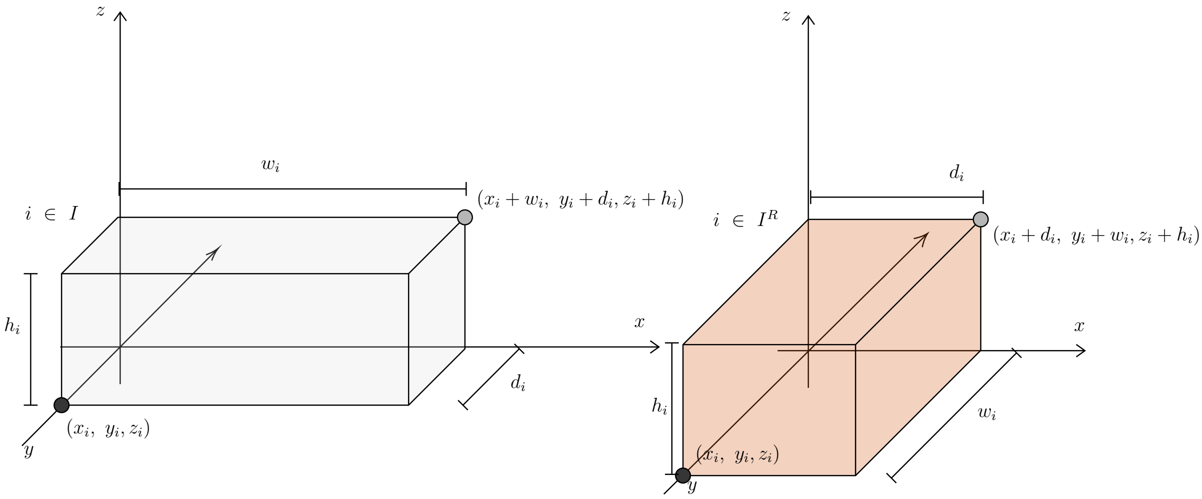


Figure 3.3: Coordinate system representation for a generic item i and its rotated clone $i \in I^R$

$$\min \quad \sum_{b \in B} (Hv_b + z_b^{max}) \tag{3.2}$$

$$\text{s.t.} \quad \sum_{b \in B} u_{ib} = 1 \qquad \forall i \in I \tag{3.3}$$

$$u_{ib} \leq v_b \qquad \forall i \in I, \forall b \in B \tag{3.4}$$

$$v_b \geq v_c \qquad \forall (b, c) \in B : b < c \tag{3.5}$$

$$x_i + w_i \leq W \qquad \forall i \in I \tag{3.6}$$

$$y_i + d_i \leq D \qquad \forall i \in I \tag{3.7}$$

$$z_i + h_i \leq H \qquad \forall i \in I \tag{3.8}$$

$$z_b^{max} \geq (z_i + h_i) - H(1 - u_{ib}) \qquad \forall i \in I, \forall b \in B \tag{3.9}$$

$$(x_i + w_i) - x_j \leq W(1 - x_{ij}^p) \qquad \forall i, j \in I \tag{3.10}$$

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

$$(y_i + d_i) - y_j \leq D(1 - y_{ij}^p) \qquad \forall i, j \in I \tag{3.12}$$

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

$$(z_i + h_i) - z_j \leq H(1 - z_{ij}^p) \qquad \forall i, j \in I \tag{3.14}$$

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

$$x_{ij}^p + x_{ji}^p + y_{ij}^p + y_{ji}^p + z_{ij}^p + z_{ji}^p \geq u_{ib} + u_{jb} - 1 \qquad \forall i, j \in I, \forall b \in B \tag{3.16}$$

The objective function 3.2 seeks to minimize the number of opened bins and the maximum height at which items were placed across bins. In a single bin configuration, since all the volume mass is concentrated inside one bin, it maximizes the cage ratio. Constraint 3.3

ensures that each item is packed in one and only one bin, while constraint 3.4 ensures that items are only packed in bins used in the solution. Since the solution has lots of symmetries with respect to the number of bins, a symmetry-breaking constraint 3.5 can be added on the opening of bins to improve solve times. Each item is also ensured to be placed inside the bin thanks to eqs. (3.6) to (3.8). The value of z_b^{max} is forced to converge to the maximum height of a given bin thanks to constraint 3.9. Constraints from 3.10 to 3.15 are used to define the precedence binary variables x_{ij}^p, y_{ij}^p, z_{ij}^p over each axis as described in the problem formulation. Constraint 3.16 then ensures that if two items are in the same bin, there needs to be at least one axis with a clear order of precedence; otherwise, the two items would overlap.

3.1.1. Orthogonal rotations

Let us extend the definition of the bin packing problem without rotations with a new formulation that 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. Given the set of tuples $(i, j) \in I^{OR}$ where i is the original item with dimensions (w_i, d_i, h_i) and j is the corresponding rotated clone with dimensions $(w_j, d_j, h_j) = (d_i, w_i, h_i)$, we can now rewrite constraint 3.3 as 3.17 to force only one of them to be part of the solution.

$$\sum_{b \in B} u_{ib} + \sum_{b \in B} u_{jb} = 1 \qquad \forall (i, j) \in I^{OR} \tag{3.17}$$

3.1.2. Discrete vertical support formulation

We now extend the model to address the constraint of vertical support. In the literature, some mathematical formulations tackle the concept of area support and, in some cases, vertex support. For example, in Elhedhli et al. a SOCP formulation of the support constraint was used but was limited to the problem of spacing between layers with one of them being fixed in position relative to the other. A similar formulation would lead to a non-linear support constraint in our case. By introducing a discretization over the XY-plane a linear version of the constraint can be formulated similar to the one proposed in Kurpel et al. without the need to discretize the z-axis as well.

Let us introduce some additional parameters to the model, let $0 \leq \alpha_s \leq 1$ be the amount of area that an item needs to have supported by other items. Let β_s be the tolerance to consider one item as being close enough to support another item (as seen in fig. 3.2). In addition to the support parameters, a parameter δ, which represents the discretization unit used to partition the XY-plane, is given. Let I^B be the set of all the tuples (i, j, b) such that $(i, j) \in I \wedge i \neq j$ and $b \in B$. We can now compute a few additional parameters that we will use to reduce the number of constraints evaluated by the model. Let γ be the maximum size over a dimension on the XY-plane between all the items as eq. (3.18), and let Δ be the set of all possible distances between the origins of two items along one discretized axis as eq. (3.19).

$$\gamma = \max_{\forall i \in I} \{w_i, d_i\} \tag{3.18}$$

$$\Delta = \left[-\left\lfloor \frac{\gamma}{\delta} \right\rfloor, \left\lfloor \frac{\gamma}{\delta} \right\rfloor \right] \tag{3.19}$$

Let $O(i, j, h, k) \to \mathbb{R}^+$ be a function that computes the amount of overlap between two items $(i, j) \in I$ given the discretized distance between each other $(h, k) \in \Delta$ such that $x_j = x_i + \delta h$ and $y_j = y_i + \delta k$ which returns the area of overlap or 0 otherwise.

Additional new variables need to be added to the ones of the original model, let s_{ij} be a set of binary variables which will assume value 1 if item $i \in I$ can offer support to item $j \in I$ and 0 otherwise. A new set of binary variables z^c_{ij} will be 1 if item $i \in I$ is close w.r.t. the z-axis to item $j \in I$, which would mean that $z_j - (z_i + h_i) \leq \beta_s$, and 0 otherwise. Let us then introduce a new set of binary variables g_i which will assume value 1 if item $i \in I$ will be on the ground or 0 otherwise. A set of binary variables s^{kh}_{ijb} will assume value 1 if item $i \in I$ will receive support from item $j \in I$ and both items will be placed in bin $b \in B$ with a discretized distance of $(k, h) \in \Delta$ between each other and 0 otherwise.

Given all the additional parameters and variables introduced, we can give a new formulation of the model with the same objective function 3.2 and the constraints in section 3.1

with the addition of the following constraints:

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

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

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

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

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

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

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

$$\sum_{(k,h) \in \Delta, b \in B : O(i,j,k,h) \neq 0} s^{kh}_{ijb} \leq s_{ij} \qquad \forall (i,j) \in I \tag{3.27}$$

$$\sum_{(k,h) \in \Delta : O(i,j,k,h) \neq 0} s^{kh}_{ijb} \leq u_{ib} \qquad \forall (i,j,b) \in I^B \tag{3.28}$$

$$\sum_{(k,h) \in \Delta : O(i,j,k,h) \neq 0} s^{kh}_{ijb} \leq u_{jb} \qquad \forall (i,j,b) \in I^B \tag{3.29}$$

Constraints 3.20 and 3.21 ensure that z^c_{ij} is forced to 1 only when the distance over the z-axis between item i and item j is within the range $[-\beta_s, \beta_s]$. The value of s_{ij} is then assigned to the logical equation $z^p_{ij} \wedge z^c_{ij}$ thanks to constraints from 3.22 to 3.24. Since some items could be left out of the solution due to the formulation of orthogonal rotations, we also ensure that support can only come from placed items thanks to constraint 3.25. Constraint 3.26 ensures that g_i will assume value 1 if item i is on the ground Constraints from 3.27 to 3.29 ensure that if a discretized support decision s^{hk}_{ijb} is 1 then every subscript of that variable must be true in the non-discretized model, so item i can give discretized support to item j in in bin b if both items are are assinged to bin b and if i can give support to item j. They also force the selection of only one possible combination of $(h,k) \in \Delta$ for which i gives support to j in bin b.

We can then define a set of constraints which given a discretized placement s^{kh}_{ijb} limits the distance between i and j to a given continuous region in space delimited by a square of the dimension of our discretization unit δ. Given every tuple of possible discretized distances between items $(k,h) \in \Delta$ and every tuple of different pairs of items in the same bin $(i,j,b) \in I^B$ such they have a non-zero discretized overlap over the XY-plane

$(O(i,j,k,h) \neq 0)$. The resulting constraints are defined in eqs. (3.30) to (3.33).

$$x_j - x_i \geq \gamma k - 2W(1 - s_{ijb}^{kh}) \tag{3.30}$$
$$x_j - x_i \leq \gamma(k+1) + 2W(1 - s_{ijb}^{kh}) \tag{3.31}$$
$$y_j - y_i \geq \gamma h - 2D(1 - s_{ijb}^{kh}) \tag{3.32}$$
$$y_j - y_i \leq \gamma(h+1) + 2D(1 - s_{ijb}^{kh}) \tag{3.33}$$

We can then introduce a feasibility constraint thet ensures that every item that isn't on the ground is supported by other items placed beneath it by at least α_s times its area, which corresponds to condition 1 of the practical constraint of vertical support.

$$\sum_{(k,h)\in\Delta, b\in B, j\in I: i\neq j \wedge O(i,j,k,h)\neq 0} O(i,j,k,h) s_{jib}^{kh} \geq \alpha_s w_i d_i - w_i d_i g_i \quad \forall i \in I \tag{3.34}$$

It is noted that every combination of $(i,j,b) \in I^B$ and $(h,k) \in \Delta$ where $O(i,j,k,h) = 0$ do not contribute to the support constraint. We can omit them from the formulation of the problem to reduce the number of constraints to evaluate.

4 | Solution algorithms

This chapter describes a heuristic algorithm to solve the 3D bin packing problem with vertical support. In section 4.1 we describe the concepts which will be used in the algorithm, like the definition of a state, insertions, and the feasibility of a solution. Since the 3D-BPP is NP-Hard, an exhaustive search for a solution isn't practical, so 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 (4.2), and outputs the best scoring state based on an ordering function defined in section 4.2.1.

4.1. States

States or packings are partial solutions to the 3D-BPP. Since our heuristic is constructive by nature, the main idea of the algorithm is that by starting from a state representing an empty solution, we'll iteratively build new states that are always closer to a complete solution to the problem. Given the formal definition of the problem (3.1) we introduce a few new definitions to facilitate the algorithm's definition. First, it is helpful to define a collection of items that still need to be assigned to a particular bin; this collection would then be used to track how many items still need to be placed. Let us define the concept of unpacked items in relation to our MILP formulation.

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 are independent between states (changes to their values in state s won't affect state s'). In order to simplify the algorithm representation, rotations are handled by simply swapping the dimensions w_i and d_i of item $i \in I$ when needed. A state s can then be defined as follows:

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

Since the heuristic will open new bins when the already opened ones are full, the number of bins in each state can vary and isn't fixed as a parameter to the problem like in the MILP formulation. Thanks to the newly introduced definitions, we can 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*

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

The proposed heuristic also stores additional data for each opened bin, which will then be used by the constructive heuristic described in section 4.3. This additional information is stored in the set $s.Q$ so that each bin $b \in B$ has an associated supporting structure $q_b \in s.Q$. The collection of items placed inside a bin, for example, is one piece of data that we store in this structure. Let us then 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*

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

In addition to the set of packed items, other supporting structures are needed to facilitate checks of the problem's constraints. 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.

Both sets $q_b.J$ and $q_b.T$ contain the items packed in 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 $O(log(|q_b.J|))$.

4.1.1. AABB Tree

To determine the feasibility of a given state, checking for overlaps with items already placed is needed. Since every item is a cuboid and our problem formulation only allows for 90 deg rotations over the z-axis, each item is contained inside a bounding box, which 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 following that direction starting from the box i

If the tree is appropriately 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.

An additional opertation $AABBGetSupporting(i, \beta_s)$ was added to compute the set of supporting boxes of item i given a tolerance β_s. This was possible by only checking intersections over the XY-plane similarly to the $AABBOverlaps$ implementation and filtering each item by the distance with tolerance.

4.1.2. Feasibility

A state s is feasible if the currently packed items in each bin $b \in s.B$ aren't overlapping any other item if they are all contained inside their bin and if each item is either on the ground or satisfies at least one of the support conditions (cond. 1, cond. 2). Since the proposed heuristic is constructive, it is more convenient to define the concept of feasibility relative to a change in the state. In the heuristic, we generate new states by applying insertions starting from an initial feasible one. Let us define the concept of insertion and how an insertion is feasible..

Insertions Given a state s and $b \in s.B$, an insertion of items is a set of non-overlapping items that are placed in b and have their z_i within tolerance from 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 non-overlapping 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 4.1. *Given a state s and an insertion $p = (b, \emptyset)$ 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 a pending insertion on s'.*

We can evaluate the changes to the score of a state based on its pending insertion. In this way, we don't have to update all the structures for every evaluated state. In addition, this property let us do fewer memory clones of states that would have been discarded either way (as seen in section 4.2). We can then define an algorithm that applies a pending insertion p on a given state s 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 An insertion $p = (b, I)$ that is pending on a given state s is feasible if every inserted item $i \in p.I$ satisfies the constraint of non-overlap (3.16), the constraint of support (3.34) and if it is placed within the size of the given bin. Given an item from the set of the inserted items $i \in p.I$, and the AABB tree for bin $p.b$ in the current state $q_b.T$ as T. Let I_{support} be the set of items that could support item i when placed in the bin, which could be saved in an appropriate structure or computed through the AABB tree as defined in section 4.1.1.

Let $HasSupport(i, I_{\text{support}})$ be the function that returns true if the considered item would verify at least one of the conditions of support (1 or 2) or false otherwise. We can define a function $IsFeasible(i, I_{\text{support}}, T)$ which returns true if the insertion of i in bin b for state s is feasible and false otherwise. If every item $i \in p.I$ is feasible and every item in I isn't overlapping the others, then insertion p is feasible. In case some items in p aren't feasible we can always define a function $RemoveInfeasibleItems(p, I_{\text{support}}, T)$ which removes every unfeasible item and returns a new insertion $p' = (b, I')$ where $I' = p.I \setminus \{i \in p.I : \neg IsFeasible(i, I_{\text{support}}, T)\}$.

Checking if a state is feasible can then be done by iteratively applying all the insertions ordered by z and updating the proper trees, or starting from an already built tree and computing the set I_{support} for each item through the tree as defined in 4.1.1.

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

Observation 4.2. *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

4.1.3. State Hashing

From a given state, it's possible to apply two different sequences of insertions and end up with two states that have all the items in the same positions. This undesirable behavior was observed during our computational experiments. A hashing mechanism needs to be introduced to enable checking if two states are likely the same in constant time. In a state s we can identify a packed item $i \in I$ in a given position (x_i, y_i, z_i) with its given dimensions (w_i, d_i, z_i) in a given bin $b \in s.B$ with a non-commutative hashing function $hash_{nc}$. The resulting hash $hash_{ib} = hash_{nc}(b, x_i, y_i, z_i, w_i, d_i, h_i)$ can identify every similar packing of an item of the same shape in that specific bin spot. Since $hash_{ib}$ identifies one item with the shape of i in the same spot as i, we can use a commutative function to combine every hash for every packed item in every bin to ignore the order with which items were added to the solution. The combined hash can then be saved inside the state structure as follows.

$$s.hash = \sum_{b \in s.B} \sum_{i \in q_b.J} hash_{ib} \tag{4.2}$$

In our tests, by filtering states with the proposed hash as seen in algo. 2, with a simple 64-bit hashing function, this mechanism allowed us to filter out almost all equal states between iterations with a low amount of collisions. Since the combining of hashes is a simple sum with modulus, the hashing of the state can also be kept updated in constant time at each iteration by simply adding the inserted hashes in the *Commit* function (algo. 1).

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 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. As noted in section 4.1.3, since by evaluating different insertions on different states it is possible to end up having two equal states, a filtering mechanism should be introduced. During each iteration, it is possible to keep the hashes of the best scoring selected states in a HashSet and discard new states with the same hash.

Given a set of initial states S^0 and the number of best states to expand at each iteration k, the described BS can be represented by algorithm 2. As observed in observation 4.2, it's possible to start the search from $S^0 = \{s_e\}$.

Algorithm 2: Beam search

```
input  : S^0, k
output: s_best
S^t ← S^0
S_final ← ∅
repeat
    S^{t+1} ← Expand(S^t) (algo. 3)
    S_final ← S_final ∪ {s ∈ S^{t+1} : IsFinal(s)} (def. 4.2)
    S^{t+1} ← S^{t+1} \ S_final
    S^{t+1} ← Sort(S^{t+1}) (sec. 4.2.1)
    S^t ← ∅
    i ← 0
    seen ← ∅
    forall s ∈ S^{t+1} do
        if s.hash ∈ seen then
            continue
        end
        S^t ← S^t ∪ Commit(s) (algo. 1)
        seen ← seen ∪ {s.hash}
        i ← i + 1
        if i > k then
            break
        end
    end
until S^t ≠ ∅
S_final ← 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 at the lowest possible height for items that still need to be packed. To reduce the number of possible expansions to evaluate we limit the search to insertions of items with unique shapes. With a similar concept to the one used in section 4.1.3, an hash for each item's dimensions can be computed on the fly or pre-computed as a problem's parameter. Given each item's hash we can then group items that have the same shape. The evaluation of new insertions can then be done with two different approaches:

- **Single Placement**: where we evaluate only the possible insertion of a single item per item type, which would generate insertions of at most 1 item,
- **Group by Hash**: where we evaluate the biggest possible insertion of a group of items of the same shape, which would generate insertions of at most the size of the group of items with the same shape.

Creating insertions of groups of similar items is usually used in Pallet Loading Problems (as for ex. [4]) to create better bases of support for upper layers. With a similar intuition, the idea of placing groups of items of the same shape is to facilitate the creation of uniform planes (not necessarily layers) to use for further insertions.

Given a set of items I and a tolerance β_s we can introduce an algorithm to group them by their shape 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. 4. 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 4.1). The described procedure is detailed in algo. 3, which can be modified with minor changes to limit the number of items to consider when in single placement mode.

Algorithm 3: Expand

```
input  : S
output: S_new
forall s ∈ S do
    S_new ← ∅
    G ← GroupByHash(s.U) (algo. 4)
    placed ← false
    forall (h, I) ∈ G do
        forall q_b ∈ s.Q do
            P ← SPBestInsertion(q_b, I) (algo. 5)
            if P ≠ ∅ then
                placed ← true
                forall p ∈ P do
                    S_new ← S_new ∪ Next(s, p) (def. 4.5)
                end
            end
        end
    end
    if placed = false then
        // Open a new bin with index |s.B| (oss. 4.1)
        S_new ← S_new ∪ Next(s, (|s.B|, ∅))
    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.

Definition 4.6. *Let $f_1(s), f_2(s), f_i(s), \ldots, f_n(s)$ be objective functions ordered by prece-*

Algorithm 4: 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
```

dence based on index $i \in \mathbb{Z}$, then

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

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 (eq. (3.1)) between bins.

4.3. Support Planes

Support Planes (SP) is a constructive heuristic based on an underlying 2D-BPP heuristic which is used to evaluate feasible insertions inside a bin starting from a set of items to pack. Since insertions must be feasible, SP maintains an internal structure to facilitate feasibility checks. The idea at the base of SP is to build a solution to the 3D-BPP by filling 2D planes called *support planes*.

Each support plane is characterized by the tuple $(z, I_{support}, I_{upper})$ where

- z: is 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.

Every item placed in the bin can either generate a new support plane or be part of the supporting items of other planes. Items placed above a particular plane, such that $z_i + h_i > z$, are considered obstacles and are added to the I_{upper} set. When evaluating a new possible insertion, given a set of items to place I, SP selects the first feasible insertion starting from the lowest plane by using a modified version of Extreme Point in two dimensions (introduced in [2]). Once no more insertions can be made on the lowest available support plane, it's removed from the set of planes. Since insertions always happen in the lowest possible planes, the set of obstacles of planes where new insertions can be made is composed of items that only have their top face above the z of the evaluated plane, such that $z_i < z < z_i + h_i$.

The Extreme Point (EP) algorithm evaluates the placement of rectangles in a plane based on a set of reference points. Each rectangle placement generates a new set of reference points which are usually introduced based on the projection of its extreme points along each axis. The extreme points of an added rectangle r placed in (x_r, y_r) of dimensions (w_r, d_r) are the top left corner $(x_r, y_r + d_r)$ and the bottom right corner $(x_r + w_r, y_r)$. In our version of the algorithm, the extreme points of each item are introduced without projection to increase the likelihood of evaluating placements that verify the support constraint. If placements in a given reference point cannot be made, items can be rotated along their z-axis if they fit. When a reference point is used for a placement, it is then removed from the pool of reference points.

Since reference points are usually ordered based on the euclidean distance from the bottom left corner of the plane and the extreme points are usually generated and projected towards the origin of each axis, the placements over one plane are usually biased towards the bottom left corner. To address the problem, we evaluate four instances of EP where each has a different coordinate change applied to every item that moves the plane's origin to each corner of the bin. This addition is based on similar approaches from the literature where it is usually used to more uniformly fill the space (ex. Gajda et al.) and was verified to yield better cage ratio results in our internal testing.

Elements from the collection of obstacles for a plane are considered fixed placements already made with their extreme points already added to the reference point collection; this allows us to evaluate placements near other items from different planes. When checking for possible placements in a reference point, the feasibility check defined in section 4.1.2 can be used to avoid selecting unfeasible insertions. A graphical representation of a support plane is shown in **??**. When a bin is opened the only support plane available is the one on the ground.

Given a function to check the feasibility of an insertion, and the function $ComparePacking(p, p')$ which defines a ranking over insertions in the same plane, the SP algorithm can be written as algorithm 5.

Algorithm 5: 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 6)
    $P \leftarrow CoordinateChange(change, P')$
    $P \leftarrow \{i \in P : IsFeasible(i, bin, I_{support}, I_{upper}, aabb)\}$
    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 insertions 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 insertion can be computed following algorithm 6

Algorithm 6: 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. 3) 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 7. This new algorithm can be used as the last step of the *Commit* algorithm for each $b \in s'.B$.

Algorithm 7: SP Apply and Filter

```
input  : s, p, z, z_min, β_s
output: s
q_b ← (q_i ∈ s.Q : i = p.b)
//Filter bad planes
q_b.Z ← q_b.Z \ {(z', I_support, I_upper) ∈ q_b.Z : z' < z_min}
//Apply insertion
forall i ∈ p.I do
    q_b.T ← InsertAABB(i, q_b.T) //If balanced O(log(n))
    generate ← true
    forall (z', I_support, I_upper) ∈ q_b.Z do
        //Based on the distance from the top of the item
        dz ← z' − (z_i + h_i)
        if 0 ≤ dz ≤ β_s then
            generate ← false
            I_support ← I_support ∪ i
        end
        else if dz < 0 then
            I_upper ← I_upper ∪ i
        end
    end
    if generate then
        q_b.Z ← q_b.Z ∪ (z_i + h_i, {i}, ∅)
    end
end
return s
```

4.3.1. Scoring Insertions

5 | Computational results

In this chapter, in section 5.1, we evaluate the proposed heuristic against the MILP model (3.1), and in section 5.2 against other heuristics from the literature. We then show the effectiveness of our approach for our case study in section 5.3. All the tests were run on a desktop computer with an AMD Ryzen-7 5800x processor with 8 cores at 3.8 GHz and 32GB of DDR4 system RAM with Windows 10. The algorithm was implemented in Java 11, and the model was run using the python APIs from CPLEX Optimization Studio 22.1.0. In every test, CPLEX was used with a maximum runtime of 1 hour. Each evaluation against the heuristic lists both operational modes described in section 4.2 listed as "Group By Hash" and "Single Placement". All the instances used in each section of this chapter are available at `https://github.com/artumino/BinPackingThesis/tree/main/tests/instances`. Out of the 100 instances used for our case study experiments, only 80 were freely sharable with the generation procedure also described in section 5.3.

5.1. Model validation

We compared our heuristic to the proposed MILP model of section 3.1 with a single bin and with no limit on the height of the bin (also referred to as the 3D strip packing problem). The heuristic was configured to run without vertex support, using only area support rules for its feasibility checks, and k was set to 200. The configured parameters for the test were $\alpha_s = 0.7$, $\beta_s = 5$, and the discretization unit for the model was $\delta = 10$. Tests were run on the first instance of the class 1 problems from the literature tests described in section 5.2 which has a bin base of 100×100. The test was run with an iterative approach by selecting only a limited amount of items from the selected instance, starting from 1 item and increasing the number of items to pack by one at each iteration. The problem created with each iteration was saved as a test instance in the same format as the one used for literature tests. A python script then loaded each generated instance sequentially and evaluated the solutions from both the MILP problem and the heuristic. Starting from instance 6, the solution from the previous instance was used to mip start the new one leading to lower execution times. Table 5.1 shows the obtained $z_{\max}$ value of the heuristic and the MILP solution, the runtime in seconds, and the number of items. Since the underlying problem is NP-Hard, it is shown that starting from instances of size bigger than 8 items; the MILP model becomes too slow for practical use while our heuristic maintains negligible execution time. Due to discretization errors, some of the model instances gave solutions that didn't have the expected amount of support and were marked with an asterisk. The solution to instance number 5 and instance number 7 is also shown in fig. 5.1.

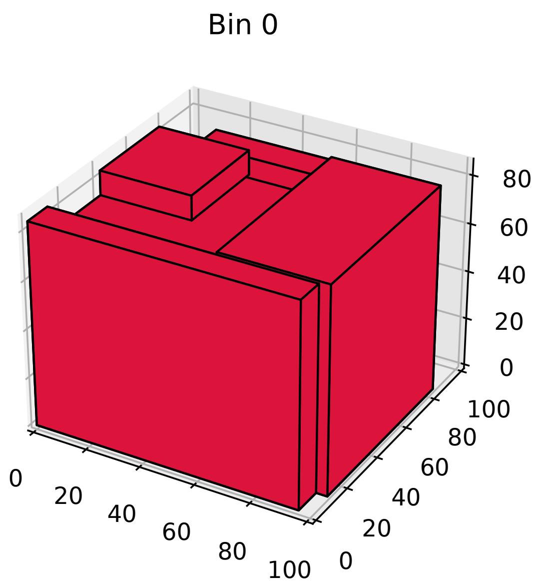
Table 5.1: Comparison with MILP model on limited set of boxes

N	Group By Hash		Single Placement		MILP Model	
	Max Z	TT(s)	Max Z	TT(s)	Max Z	TT(s)
1	85	0.00	85	0.00	85	0.01
2	85	0.00	85	0.00	85	0.09
3	85	0.00	85	0.00	85	0.24
4	85	0.01	85	0.01	85	0.64
5	85	0.02	85	0.02	85	16.48
6	158	0.06	158	0.05	158*	594.32
7	158	0.07	158	0.08	158*	3,178.00
8	160	0.10	160	0.08	-	-
9	169	0.09	161	0.10	-	-
10	218	0.12	218	0.13	-	-
11	240	0.12	240	0.12	-	-
12	310	0.13	316	0.16	-	-
13	310	0.15	333	0.18	-	-
14	310	0.20	333	0.22	-	-
15	406	0.21	397	0.27	-	-
16	435	0.23	452	0.36	-	-
17	429	0.27	515	0.41	-	-
18	432	0.32	522	0.47	-	-
19	458	0.35	522	0.55	-	-
20	539	0.37	564	0.62	-	-

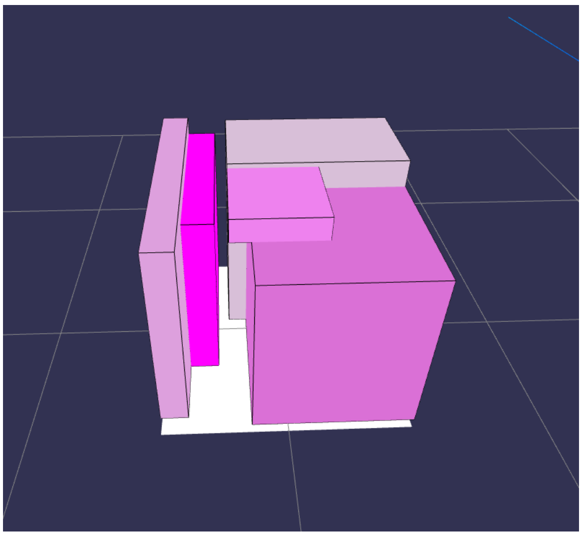
* Some boxes had lower support than expected due to discretization errors within the $0.65 \leq \alpha_s \leq 0.7$ range.

5.2. Literature results

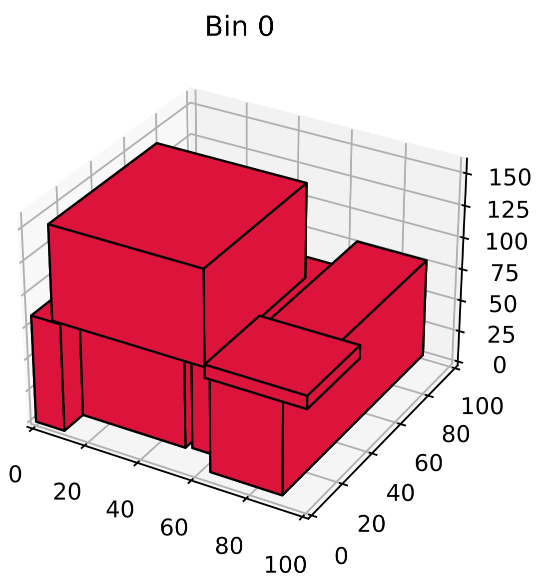
The heuristic was also evaluated against instances from the literature defined by Martello et al.. Since these instances were designed for heuristics without the vertical support constraint and orthogonal rotations, we ran the experiments with a relaxed version of our constraints. The heuristic was configured to ignore the support constraint with $\alpha_s = 0$



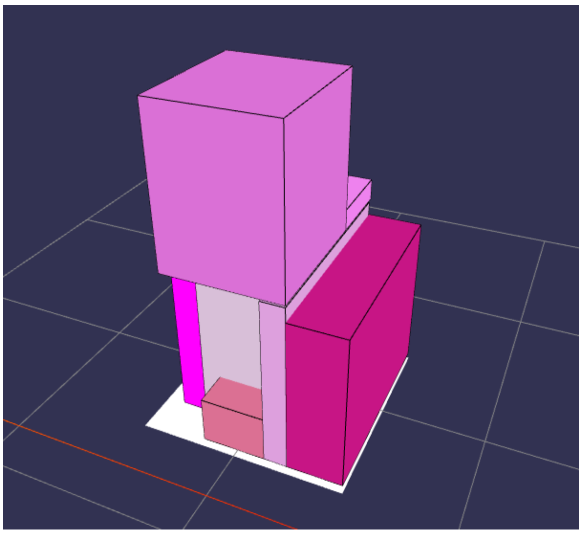
(a) MILP, Instance 5



(b) Heuristic k=200, Instance 5



(c) MILP, Instance 7



(d) Heuristic k=200, Instance 7

Figure 5.1: Graphical comparison between solutions from the heuristic and from the MILP model

and $\beta_s = 1$. We also disabled orthogonal rotations and stopped scoring insertions based on the support area available (as described in section 4.3.1).

The literature instances are divided into classes from 1 to 8, with each class having a different bin size and various distributions of types of items. Instances were generated with the C++ instance generator provided by Martello et al. which allows the generation of problem instances given a problem class and the number of items to use. We generated 10 instances for each pair of problem class and number of items $n \in \{50, 100, 150, 200\}$ for a total of 320 instances.

In table 5.2 we compare the average number of opened bins across 10 instances for each problem class and n number of items combination. The results are then compared to the most effective methods from the literature ordered by publishing date and listed as TS3 [10], GLS [5], GASP [3], GVND [13], EHGH2 [8], BRKGA [7], BRKGA-VD [14]. The best values of all the heuristics are marked in bold. Best scoring values across different configurations of our heuristic are marked in italic instead. Results show an average gap of 4.1% compared to the average value across the other heuristics and an average gap of 5.32% with respect to the best performing one.

In table 5.3 we give an approximate comparison between the average execution time of our heuristic with respect to BRKGA-VD. Execution times for BRKGA-VD were normalized by comparing directly the floating-point operations per second of the processors used, which resulted in dividing BRKGA-VD execution times by a normalization term of 9.3. The times are averaged across all 8 classes of problems based on the size of the instance. In the last column, we also included the average gap of each configuration of the heuristic with respect to the values of BRKGA-VD.

5.3. Case study results

Case study experiments were conducted on a series of problem instances that were divided between 20 real-world instances and 80 generated instances composed of items sampled from a population of real-world products. Each instance was anonymized and converted to a format similar to the one used for the literature tests thanks to a Rust program available at `https://github.com/artumino/BinPackingThesis/tree/main/additional/testConverter`. Support parameters for the heuristic were set to $\alpha_s = 0.7$ and $\beta_s = 10$ with both area and vertex support enabled. All dimensions of the bin, items, and tolerances are assumed to be in millimeters. Different values of $k \in \{1, 5, 10, 20, 50, 100, 200\}$ were tested as well as both placement modes.

Table 5.2: Literature results for k=50

Class	n	Group by Hash k=50	Single Placement k=50	TS3	GLS	GASP	EHGH2	GVN	BRKGA	BRKGA-VD
1	50	14.10	*14*	**13.4**	**13.4**	**13.4**	13.8	**13.4**	**13.4**	**13.4**
	100	28	*27.7*	**26.6**	**26.6**	26.9	27.6	**26.6**	**26.6**	**26.6**
	150	38.4	*37.9*	36.7	37	37	39.8	36.4	36.4	**36.3**
	200	53	52.7	51.2	51.2	51.6	**50.6**	50.9	50.8	50.8
2	50	*14.6*	14.8	**13.8**	-	-	-	**13.8**	**13.8**	**13.8**
	100	26.6	26.7	25.7	-	-	-	25.7	25.6	**25.5**
	150	38.3	39	37.2	-	-	-	36.9	**36.6**	**36.6**
	200	*51*	51.7	50.1	-	-	-	**49.4**	**49.4**	**49.4**
3	50	13.9	13.9	**13.3**	-	-	-	**13.3**	**13.3**	**13.3**
	100	27.8	27.3	26	-	-	-	26	**25.9**	**25.9**
	150	39.2	39	37.7	-	-	-	37.6	**37.5**	**37.5**
	200	51.8	*51.2*	50.5	-	-	-	50	**49.8**	**49.8**
4	50	*29.7*	*29.7*	**29.4**	**29.4**	**29.4**	**29.4**	**29.4**	**29.4**	**29.4**
	100	*59.2*	*59.2*	59	59	59	59.5	59	59	**58.9**
	150	*87.6*	87.7	**86.8**	**86.8**	**86.8**	90.4	**86.8**	**86.8**	**86.8**
	200	*119.5*	*119.5*	**118.8**	119	**118.8**	119	**118.8**	**118.8**	**118.8**
5	50	*8.6*	*8.6*	8.4	8.3	8.4	**7.9**	8.3	8.3	8.3
	100	16	*15.6*	15	15.1	15.1	**14.6**	15	15	15
	150	21.7	21.4	20.4	20.2	20.6	21.5	20.4	20.1	**19.9**
	200	29	28.4	27.6	27.2	27.7	29.6	**27.1**	**27.1**	**27.1**
6	50	*10*	10.3	9.9	9.8	9.9	11.8	9.8	**9.7**	**9.7**
	100	19.8	19.7	19.1	19.1	19.1	19.2	19	**18.9**	**18.9**
	150	30.3	30.2	29.4	29.4	29.5	29.8	29.2	**29**	**29**
	200	38.9	*38.5*	37.7	37.7	38	38.7	37.4	**37.3**	**37.3**
7	50	7.8	*7.6*	7.5	**7.4**	7.5	**7.4**	**7.4**	**7.4**	**7.4**
	100	*13.2*	*13.2*	12.5	12.3	12.7	13.5	12.5	**12.2**	**12.2**
	150	17.1	16.8	16.1	15.8	16.6	18.2	16	15.3	**15.2**
	200	24.9	24.7	23.9	23.5	24.2	24.1	23.5	**23.4**	**23.4**
8	50	9.9	9.7	9.3	**9.2**	9.3	9.4	**9.2**	**9.2**	**9.2**
	100	*19.6*	20	18.9	18.9	19	18.9	18.9	18.9	**18.8**
	150	25.7	25.8	24.1	23.9	24.8	26	24.1	**23.6**	**23.6**
	200	31.6	*31.2*	30.3	29.9	31.1	35.8	29.8	**29.3**	**29.3**

Table 5.3: Average execution time of literature results with bin gap

Heuristic		Execution Time (s)				Bin Gap %
		n=50	n=100	n=150	n=200	
Group By Hash	k=1	0.03	0.11	0.28	0.54	5.82
	k=5	0.08	0.38	1.00	2.09	5.56
	k=10	0.15	0.73	1.93	4.00	5.54
	k=20	0.29	1.40	3.77	7.71	5.30
	k=50	0.70	3.50	9.39	19.59	5.19
Single Placement	k=1	0.05	0.18	0.50	1.05	5.61
	k=5	0.12	0.72	2.10	4.62	5.26
	k=10	0.23	1.38	4.11	8.95	5.19
	k=20	0.46	2.67	8.21	17.64	4.98
	k=50	1.12	6.45	20.39	43.50	4.75
BRKGA-VD		1.85	8.69	20.53	39.85	0.00

Each generated instance is composed of a random number of N items sampled from a given range of possible instance sizes. All generated instances had a bin of standard size $800 \times 1200 \times 2000$. We identified four ranges of interest and generated 20 instances for each range as follows:

- **Class 1-20**: a class of instances with the target sizes for our case-study $N \in [70, 100]$
- **Class 21-40**: a class of small sized instances with number of items $N \in [50, 70]$
- **Class 41-60**: a class of medium sized instances with number of items $N \in [70, 120]$
- **Class 61-80**: a class of big instances with number of items $N \in [120, 200]$

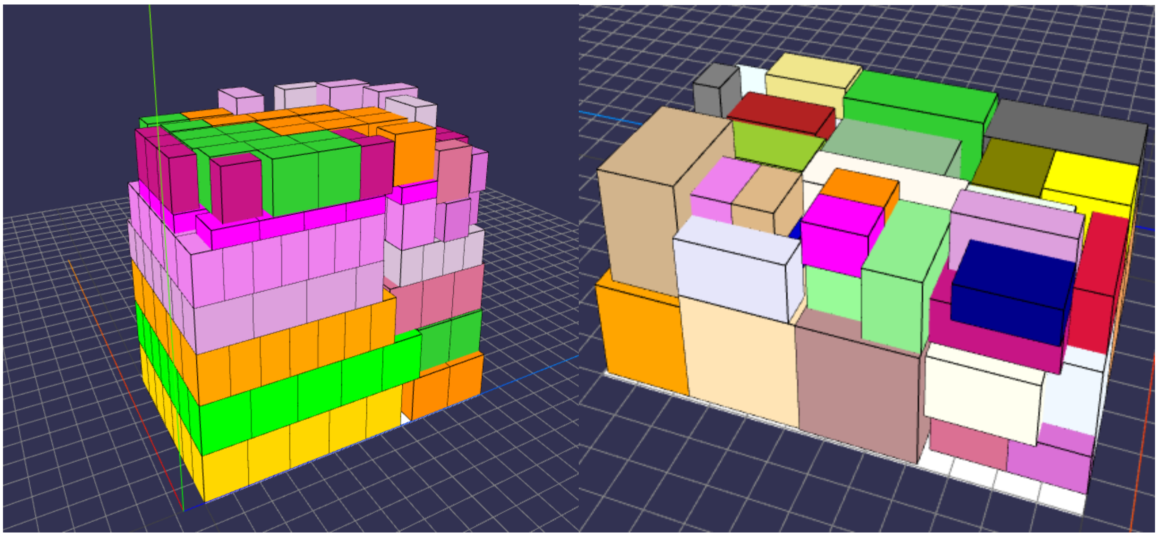
Given an input N (the size of the test instance), the generation procedure uniformly sampled an item type from a population of real-world products. The number of items of that type to add to the test instance was then sampled from a normal distribution $\mathcal{N}(\mu = 4.6, \sigma = 1.8)$, floored to be an integer value and clamped to avoid generating more items than N. This uniform sampling of item types was done until the instance was composed of N items.

Real-world instances are listed as **Class 81-100** and have a variable number of items between $[25, 345]$, a variable bin size (although similar to the one used for the generated instances), and a variable number of items of the same type. Some instances were homogeneous with only a few unique items, and some were heterogeneous with every item of a different type. An example of real-world instances is shown in fig. 5.2 where items of the same shape are marked with the same color.

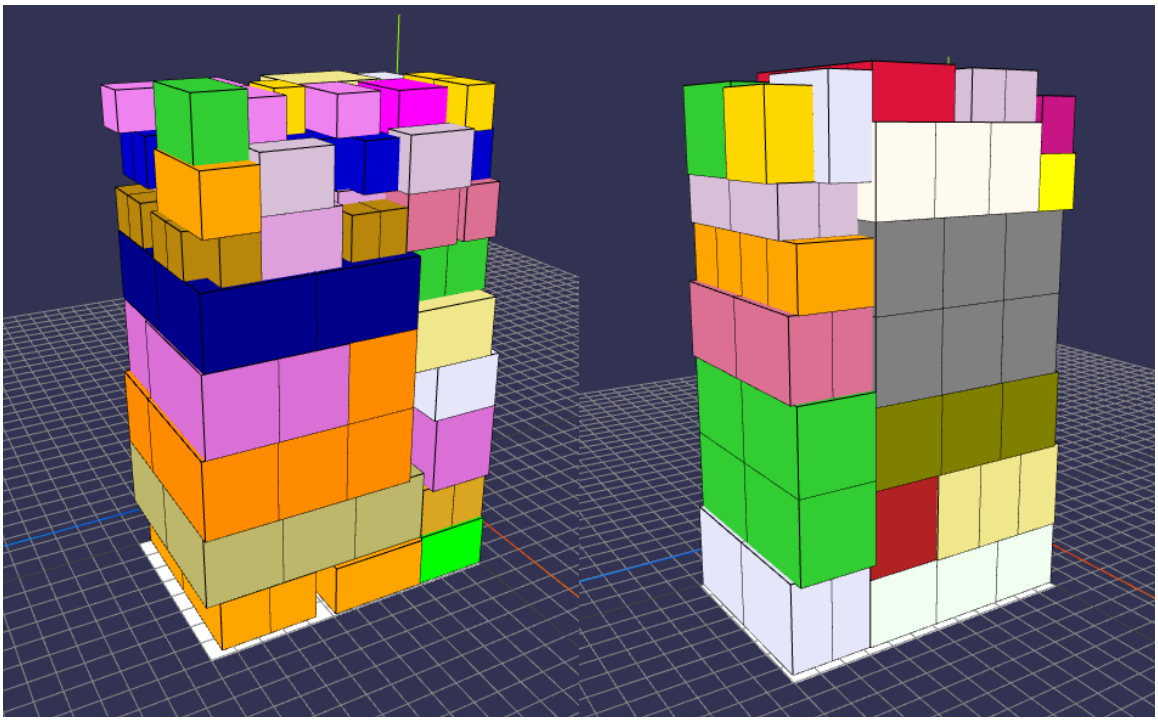
Table 5.3 shows the average results over the 20 instances per class, divided by each configuration of the heuristic with different values of k. The results shown include the total execution time in milliseconds (TT), the number of opened bins (B), and the average cage ratio between the opened bins (CR). It is clear that although the "Single Placement" method had better results when dealing with a relaxed version of the problem, grouping items by type shows considerable improvements under all measured metrics when taking vertical support into account. Most of the configurations lead to an average cage ratio of more than 70%, which was the target value for our case study. It is also possible to see that increasing the value of k improves the quality of the solutions, on average, at the expense of a higher execution time. By doing a case-by-case analysis of each experiment, we discovered that increasing k can temporarily worsen the solution in some instances. A further study of the problematic instances highlighted that the current greedy scoring mechanism of the states leads to cutting out good solutions too early. Further improvements are considered in chapter 6.

Table 5.4: Summary of use-case tests

Instance		Single Placement			Group 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 1-20	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 21-40	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 41-60	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 61-80	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 81-100	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**



(a) Instance 95 (b) Instance 82



(c) Instance 56 (d) Instance 66, Bin 1

Figure 5.2: Solutions of case study tests with the "Group by Hash" placement and $k = 200$

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.

[2] T. G. Crainic, G. Perboli, and R. Tadei. Extreme point-based heuristics for three-dimensional bin packing. *Informs Journal on computing*, 20(3):368–384, 2008.

[3] T. G. Crainic, G. Perboli, and R. Tadei. Ts2pack: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research*, 195(3):744–760, 2009.

[4] S. Elhedhli, F. Gzara, and B. Yildiz. Three-dimensional bin packing and mixed-case palletization. *INFORMS Journal on Optimization*, 1(4):323–352, 2019.

[5] O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for the three-dimensional bin-packing problem. *Informs journal on computing*, 15(3):267–283, 2003.

[6] M. Gajda, A. Trivella, R. Mansini, and D. Pisinger. An optimization approach for a complex real-life container loading problem. *Omega*, 107:102559, 2022. ISSN 0305-0483. doi: https://doi.org/10.1016/j.omega.2021.102559. URL `https://www.sciencedirect.com/science/article/pii/S0305048321001687`.

[7] J. F. Gonçalves and M. G. Resende. A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145 (2):500–510, 2013.

[8] M. Hifi, S. Negre, and L. Wu. Hybrid greedy heuristics based on linear programming for the three-dimensional single bin-size bin packing problem. *International Transactions in Operational Research*, 21(1):59–79, 2014.

[9] D. V. Kurpel, C. T. Scarpin, J. E. P. Junior, C. M. Schenekemberg, and L. C. Coelho. The exact solutions of several types of container loading problems. *European Journal of Operational Research*, 284(1):87–107, 2020.

[10] A. Lodi, S. Martello, and D. Vigo. Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research*, 141(2):410–420, 2002.

[11] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations research*, 48(2):256–267, 2000.

[12] C. Paquay, M. Schyns, and S. Limbourg. A mixed-integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application. *International Transactions in Operational Research*, 23(1-2):187–213, 2016.

[13] F. Parreño, R. Alvarez-Valdés, J. F. Oliveira, and J. M. Tamarit. A hybrid grasp/vnd algorithm for two-and three-dimensional bin packing. *Annals of Operations Research*, 179(1):203–220, 2010.

[14] A. Zudio, D. H. da Silva Costa, B. P. Masquio, I. M. Coelho, and P. E. D. Pinto. Brkga/vnd hybrid algorithm for the classic three-dimensional bin packing problem. *Electronic Notes in Discrete Mathematics*, 66:175–182, 2018.

A | Appendix A

Table A.1: Case study results 1-5

Instance		Single Placement			Group 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: Case study results 6-10

Instance		Single Placement			Group 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: Case study results 11-15

Instance		Single Placement			Group 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: Case study results 16-20

Instance		Single Placement			Group 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: Case study results 21-25

Instance		Single Placement			Group 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: Case study results 26-30

Instance		Single Placement			Group 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: Case study results 31-35

Instance		Single Placement			Group 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: Case study results 36-40

Instance		Single Placement			Group 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: Case study results 41-45

Instance		Single Placement			Group 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: Case study results 46-50

Instance		Single Placement			Group 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: Case study results 51-55

Instance		Single Placement			Group 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: Case study results 56-60

Instance		**Single Placement**			**Group 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: Case study results 61-65

Instance		Single Placement			Group 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: Case study results 66-70

Instance		Single Placement			Group 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: Case study results 71-75

Instance		Single Placement			Group 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: Case study results 76-80

Instance		Single Placement			Group 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: Case study results 81-85

Instance		Single Placement			Group 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: Case study results 86-90

Instance		Single Placement			Group 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: Case study results 91-95

Instance		Single Placement			Group 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: Case study results 96-100

Instance		Single Placement			Group 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

Table A.21: Summary of opened bins by heuristic

Class		Group by Hash					Single Placement					TS3	GLS	GASP	EHGH2	GVN	BRKGA	BRKGA-VD
Instance	n	k=1	k=5	k=10	k=20	k=50	k=1	k=5	k=10	k=20	k=50							
1	50	14.5	14.5	14.5	14.5	14.10	14.4	14.3	14.3	14.4	14	**13.4**	**13.4**	**13.4**	13.8	**13.4**	**13.4**	**13.4**
	100	28.4	28.3	28.3	28.4	28	27.7	27.8	27.8	27.9	27.7	**26.6**	**26.6**	26.9	27.6	**26.6**	**26.6**	**26.6**
	150	38.5	38.5	38.4	38.3	38.4	38.4	38.5	38.5	38.3	37.9	36.7	37	37	39.8	36.4	36.4	**36.3**
	200	53.1	53.2	53	52.9	53	52.7	52.7	52.6	52.6	52.7	51.2	51.2	51.6	**50.6**	50.9	50.8	50.8
2	50	14.8	14.7	14.7	14.7	14.6	14.9	14.8	14.8	14.9	14.8	**13.8**	-	-	-	**13.8**	**13.8**	**13.8**
	100	26.5	26.6	26.6	26.6	26.6	27.1	26.9	26.9	26.7	26.7	25.7	-	-	-	25.7	25.6	**25.5**
	150	38.4	38.3	38.2	38.3	38.3	39.2	39.2	39.1	39.1	39	37.2	-	-	-	36.9	**36.6**	**36.6**
	200	51.3	51.2	51.1	51.3	51	51.9	51.8	51.9	51.8	51.7	50.1	-	-	-	**49.4**	**49.4**	**49.4**
3	50	14.2	14.1	13.9	13.8	13.9	14	14.1	13.9	14	13.9	**13.3**	-	-	-	**13.3**	**13.3**	**13.3**
	100	27.7	27.7	27.7	27.5	27.8	27.6	27.5	27.2	27.3	27.3	26	-	-	-	26	**25.9**	**25.9**
	150	39.4	39.4	39.5	39.5	39.2	39.1	38.9	38.9	39	39	37.7	-	-	-	37.6	**37.5**	**37.5**
	200	51.8	51.8	51.7	51.6	51.8	51.4	51.4	51.4	51.3	51.2	50.5	-	-	-	50	**49.8**	**49.8**
4	50	29.7	29.7	29.7	29.7	29.7	29.7	29.7	29.7	29.7	29.7	**29.4**	**29.4**	**29.4**	**29.4**	**29.4**	**29.4**	**29.4**
	100	59.2	59.2	59.2	59.2	59.2	59.2	59.2	59.2	59.2	59.2	59	59	59	59.5	59	59	**58.9**
	150	87.6	87.6	87.6	87.6	87.6	87.6	87.7	87.6	87.6	87.7	**86.8**	**86.8**	**86.8**	90.4	**86.8**	**86.8**	**86.8**
	200	119.5	119.5	119.5	119.5	119.5	119.5	119.5	119.5	119.5	119.5	**118.8**	119	**118.8**	119	**118.8**	**118.8**	**118.8**
5	50	8.7	8.7	8.7	8.7	8.6	8.7	8.7	8.6	8.6	8.6	8.4	8.3	8.4	**7.9**	8.3	8.3	8.3
	100	16	16	16	16	16	15.7	15.6	15.6	15.6	15.6	15	15.1	15.1	**14.6**	15	15	15
	150	21.6	21.6	21.6	21.7	21.7	21.3	21.3	21.3	21.3	21.4	20.4	20.2	20.6	21.5	20.4	20.1	**19.9**
	200	28.9	28.9	28.9	29	29	28.4	28.2	28.3	28.4	28.4	27.6	27.2	27.7	29.6	**27.1**	**27.1**	**27.1**
6	50	10.5	10.3	10.1	10.2	10	10.7	10.5	10.4	10.2	10.3	9.9	9.8	9.9	11.8	9.8	**9.7**	**9.7**
	100	19.8	19.8	20	19.5	19.8	19.8	19.7	19.8	19.7	19.7	19.1	19.1	19.1	19.2	19	**18.9**	**18.9**
	150	30.4	30.4	30.3	30.3	30.3	30.4	30.3	30.3	30	30.2	29.4	29.4	29.5	29.8	29.2	**29**	**29**
	200	39.1	39.2	39.2	38.6	38.9	39.1	38.6	38.8	38.6	38.5	37.7	37.7	38	38.7	37.4	**37.3**	**37.3**
7	50	7.9	7.7	7.9	7.8	7.8	7.7	7.9	7.9	7.7	7.6	7.5	**7.4**	7.5	**7.4**	**7.4**	**7.4**	**7.4**
	100	13.5	13.2	13.4	13.3	13.2	13.5	13.5	13.5	13.4	13.2	12.5	12.3	12.7	13.5	12.5	**12.2**	**12.2**
	150	16.9	16.9	17	16.9	17.1	17.1	16.7	16.9	16.8	16.8	16.1	15.8	16.6	18.2	16	15.3	**15.2**
	200	25.1	25	25	25.2	24.9	24.9	24.5	24.8	24.7	24.7	23.9	23.5	24.2	24.1	23.5	**23.4**	**23.4**
8	50	10	10.1	9.9	9.8	9.9	9.7	9.6	9.6	9.6	9.7	9.3	**9.2**	9.3	9.4	**9.2**	**9.2**	**9.2**
	100	19.6	19.7	19.7	19.6	19.6	20	20	19.9	20	20	18.9	18.9	19	18.9	18.9	18.9	**18.8**
	150	25.8	25.6	25.6	25.7	25.7	26	25.8	25.7	25.8	25.8	24.1	23.9	24.8	26	24.1	**23.6**	**23.6**
	200	31.5	31.5	31.6	31.3	31.6	31.6	31.6	31.6	31.4	31.2	30.3	29.9	31.1	35.8	29.8	**29.3**	**29.3**

Table A.22: Detailed execution times for literature tests in milliseconds

Class		Group by Hash					Single Placement					BRKGA-VD
Instance	n	k=1	k=5	k=10	k=20	k=50	k=1	k=5	k=10	k=20	k=50	
1	50	80.50	85.20	150.70	307.00	820.30	85.00	109.40	192.80	390.60	968.20	1,293.25
	100	137.90	439.10	819.60	1,500.60	3,737.60	177.20	634.80	1,181.50	2,409.10	5,540.10	7,005.10
	150	341.50	1,237.10	2,478.60	4,677.30	12,143.30	494.10	1,865.90	3,817.80	7,625.60	18,751.70	17,674.42
	200	697.40	2,808.20	5,144.30	10,319.20	26,679.40	1,087.30	4,629.80	9,134.60	18,256.50	44,792.70	34,702.21
2	50	24.70	72.80	126.50	220.10	555.70	35.80	125.60	243.20	510.80	1,190.30	1,293.25
	100	105.80	342.50	647.50	1,528.70	3,921.30	194.00	869.00	1,724.50	3,391.50	8,489.30	7,112.88
	150	266.00	974.60	1,842.70	3,399.20	8,160.60	590.30	2,599.20	4,943.70	10,118.50	24,492.20	17,782.19
	200	532.40	1,971.00	3,817.30	7,228.50	17,694.90	1,320.50	5,792.70	11,790.20	22,535.60	53,024.40	34,486.67
3	50	28.10	82.70	156.30	302.50	710.20	32.50	112.00	204.00	401.30	982.40	1,293.25
	100	120.10	489.30	910.70	1,625.50	4,313.90	162.20	621.70	1,231.80	2,234.10	6,042.70	6,897.33
	150	341.20	1,263.80	2,455.60	4,719.40	11,596.80	461.40	1,979.50	4,027.40	8,214.90	20,657.80	18,428.81
	200	690.00	2,694.00	5,307.70	10,418.30	25,865.20	1,018.20	4,318.50	8,788.70	17,801.60	44,903.80	35,672.15
4	50	45.10	129.40	255.60	506.10	1,200.90	95.60	150.20	280.40	532.50	1,377.00	1,185.48
	100	220.70	817.40	1,582.80	2,973.30	7,125.80	246.00	945.50	1,742.80	3,532.90	8,636.60	6,574.02
	150	559.70	2,074.50	3,944.40	7,778.10	19,574.70	644.20	2,601.60	5,089.70	9,968.20	24,431.50	17,135.56
	200	1,143.20	4,685.50	8,938.80	17,291.40	44,421.60	1,415.90	6,027.40	11,581.60	23,829.20	58,029.70	34,271.13
5	50	23.40	75.40	152.80	275.10	701.90	40.60	155.90	318.80	631.80	1,617.20	3,233.13
	100	69.80	230.80	438.70	861.00	2,229.90	212.40	937.60	1,703.40	3,547.50	7,580.70	13,040.27
	150	164.90	596.50	1,134.80	2,415.50	5,727.40	618.10	2,532.10	4,824.10	9,131.20	27,346.00	29,960.29
	200	283.80	1,055.90	2,071.00	3,899.90	9,916.70	1,210.60	4,988.80	9,872.90	20,611.90	56,078.50	57,226.32
6	50	22.40	73.20	136.10	262.90	568.60	26.90	88.80	177.70	349.40	836.40	969.94
	100	88.50	315.00	583.50	1,139.40	2,684.80	136.40	555.30	1,030.40	1,839.80	4,456.70	5,065.23
	150	223.40	796.90	1,448.70	3,019.10	7,749.50	323.80	1,360.10	2,561.50	5,367.50	11,886.10	13,686.90
	200	401.90	1,555.60	2,820.30	5,229.50	12,899.70	624.40	2,523.20	4,943.60	9,218.90	23,577.80	27,697.11
7	50	18.00	53.70	103.60	190.80	487.40	32.80	113.30	218.20	453.50	1,105.60	2,586.50
	100	68.30	229.50	433.30	846.80	1,993.60	154.30	656.70	1,255.30	2,261.20	5,893.50	10,669.31
	150	153.10	547.60	1,164.10	2,190.90	5,292.60	413.90	1,894.10	3,777.40	7,156.60	17,704.60	24,571.75
	200	270.00	976.90	2,056.00	3,804.10	10,078.00	808.70	4,513.50	7,854.10	14,518.60	32,642.10	46,125.92
8	50	18.60	60.40	119.30	220.30	528.50	29.50	109.00	200.90	406.60	919.30	2,909.81
	100	65.00	208.00	395.90	758.20	1,983.70	160.20	576.30	1,179.10	2,155.50	4,981.50	13,148.04
	150	151.70	516.30	937.80	1,970.20	4,837.30	433.10	1,964.40	3,815.50	8,109.50	17,854.20	25,002.84
	200	296.80	1,004.90	1,835.80	3,471.70	9,174.80	896.00	4,149.50	7,612.00	14,338.20	34,916.60	48,604.65

Table A.23: Cage ratio of literature tests

Class		**Group by Hash**					**Single Placement**				
Instance	n	k=1	k=5	k=10	k=20	k=50	k=1	k=5	k=10	k=20	k=50
1	50	69.36	69.84	69.66	69.74	71.73	70.09	70.5	70.45	70.06	**72.36**
	100	73.01	73.62	73.37	73.44	**74.78**	74.56	74.25	74.26	74.17	74.56
	150	75.66	75.53	75.61	75.91	76.07	75.93	75.93	76.04	76.31	**76.9**
	200	75.33	75.28	75.56	75.62	75.78	75.71	75.8	**75.99**	75.9	75.93
2	50	69.6	69.5	69.48	**69.79**	69.31	67.92	68.71	68.09	68.32	68.78
	100	73.05	72.76	**73.09**	72.7	72.7	71.37	71.74	71.74	72.22	72.54
	150	72.95	73.15	**73.28**	73.11	73.04	71.51	71.28	71.5	71.38	71.66
	200	73.21	73.29	73.34	73.24	**73.42**	72.23	72.36	72.12	72.43	72.55
3	50	71.61	71.81	73.08	72.94	72.87	72.38	71.9	72.63	71.88	**73.4**
	100	73.52	73.62	73.7	74.37	73.73	73.99	74.28	**74.97**	74.79	74.83
	150	74.94	75.1	75.04	75.04	75.78	75.6	75.98	**76.04**	75.83	75.94
	200	76.09	76.01	76.21	76.43	76.08	76.58	76.74	76.82	77.01	**77.03**
4	50	61.49	61.49	61.62	**61.8**	61.79	61.49	61.6	61.7	61.73	61.78
	100	63.48	63.46	63.53	63.53	**63.58**	63.49	63.5	63.52	63.53	63.57
	150	61.91	61.96	61.92	61.95	**61.98**	61.94	61.84	61.94	61.95	61.96
	200	61.83	**61.84**	61.82	**61.84**	61.8	61.81	61.83	61.83	61.82	61.83
5	50	69.08	68.72	69.03	69.44	70.5	69.45	69.54	70.35	70.18	**70.69**
	100	73.49	73.66	73.8	74.1	73.71	74.52	74.94	74.85	**75.02**	75.01
	150	76.38	76.48	76.5	76.14	76.43	77.32	77.28	**77.33**	77.23	77.01
	200	77.12	77.19	77.11	77.06	76.99	77.88	**78.46**	78.14	77.99	77.8
6	50	77.26	78.65	79.52	79.09	**80.12**	75.65	77.23	77.44	79.06	78.44
	100	84.63	84.59	84.17	**85.74**	85.03	84.2	84.69	84.34	84.62	84.91
	150	84.66	85.17	85.33	85.45	85.4	84.97	85.32	85.2	**86.48**	85.93
	200	86.35	86.35	86.36	87.17	86.83	85.99	87.1	86.9	87.27	**87.53**
7	50	64.49	66.32	65.35	66.5	66.69	66.8	65.44	65.53	66.87	68.5
	100	71.22	72.94	71.64	72.64	**73.37**	71.48	71.63	71.8	72.03	73.12
	150	76.65	76.63	76.31	76.69	76.05	75.11	**77.25**	76.11	76.73	76.76
	200	77.48	77.95	77.97	77.08	78.06	77.66	**79**	78.04	78.16	78.44
8	50	69.61	69.48	70.75	70.81	70.62	70.7	71.38	71.38	**71.83**	71.14
	100	74.38	**74.51**	74.38	74.44	74.28	73.16	73.68	73.43	73.13	73.3
	150	77.07	**77.52**	77.19	77.07	77.22	75.89	76.73	76.81	76.4	76.61
	200	79.5	79.51	79.24	**79.86**	79.36	78.47	78.59	78.61	78.99	79.76

Bold values are the best average values

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.