OVERLAP PACKING OPTIMIZATION FOR SPACECRAFT LAYOUT DESIGN

by

Richard Arthur Alaimo

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Engineering Management

Charlotte

2018

Approved by:

_____
Dr. Churlzu Lim

_____
Dr. Simon Hsiang

_____
Dr. Tiantian Nie

ABSTRACT

RICHARD ARTHUR ALAIMO. Overlap Packing Optimization for Spacecraft Layout Design. (Under the direction of DR. CHURLZU LIM)

Packing optimization is a common problem that is encountered on a day-to-day basis and has led to an extensive study over the years. One variant of the packing problem that is beginning to emerge is the *overlap cuboid packing problem* (OCPP), which allows for items that need to be packed to share space in order to further minimize the dimensions of the container. The objective of this study is to investigate a mathematical model, specifically a mixed-integer linear program (MILP), that can effectively provide an overlap packing solution in the context of designing the layout for a spacecraft module. For this particular layout design problem, the items that need to be packed are represented as task volumes, known as *gradient cuboids*, where astronauts perform various tasks and the container is the spacecraft module. A bi-objective function was proposed to allow the dimensions of the module to be minimized, as well as enforcing specific adjacency requirements that might exist between gradient cuboids.

Following the formulation of the model, the efficacy of the model is evaluated on test problems that consist of 20 randomly generated small-scale instances with seven cuboids and one full-scale instance with 24 cuboids. In this numerical study, each of the test instances were solved under 10 different design scenarios, which varied depending on the enforced vertical overlap and restrictions on the dimensions of the container. To obtain solutions of the MILPs, a commercial solver, Gurobi, was called from Matlab. The visualization of results demonstrates how the proposed model effectively generates layout

designs. We also observed how the proposed model can accommodate prioritization between two criteria: size of the container and enforcement of adjacency requirements.

Noting that OCPP is a generalization of the NP-hard packing problem, it is unlikely to efficiently find an optimal solution as the problem size increases. It consumed 81.02 seconds on average to solve small-scale instances. However, the solution of the full-scale problem had to be terminated after a 4-hour run, when the optimality gap was 62.53% on average for 10 layout designs. Observing the large optimality gap, we turned our attention to further enhance the model to help reduce the computational burden when finding a solution. As a result, seven additional sets of constraints were proposed.

An additional numerical study was performed where each constraint was added to the base model individually using the test problem in order to determine which constraints provide the most benefit to solving the model across a set of scenarios. Following this, combinations of constraints were created for different layout designs on the randomly generated problems to determine if further improvement was possible. It was observed that there was an average improvement of 65.12% in solution time when applied to small-scale instances. These combinations were applied to the full-scale problem, where it was observed that there was an average improvement of 14.92% in optimality gap after 4-hours.

DEDICATION

I dedicate this to the entire Alaimo family, for all past, current, and future generations.

ACKNOWLEDGEMENTS

I would like to thank the Department of Systems Engineering and Engineering Management at the University of North Carolina at Charlotte for providing me with financial support as well as the opportunity to perform research for a complex engineering problem.

I would especially like to thank Dr. Churlzu Lim, my advisor and chairman of my committee. He has had great confidence in my ability to perform research alongside him ever since undergraduate studies and has provided much needed support for succeeding in graduate school. He has also provided invaluable advice on how to succeed in my professional career, as well as in life. I would also like to thank Dr. Simon Hsiang, Professor and Department Chair of the Department of Systems Engineering and Engineering Management, for also having confidence in my ability to perform research and providing support wherever it was needed. I would also like to thank Dr. Tiantian Nie, Assistant Professor, for agreeing to serve as a committee member.

I would like to thank fellow students and collaborators who I have had the opportunity to work with for this research, as well as for other related projects. These experiences have assisted in enhancing my collaboration skills and will be useful for future endeavors.

Finally, I would like to thank my family, most importantly my parents, who have been supportive of my decision making academically and professionally to help prepare for however the future may unfold.

This research was supported by the National Aeronautics and Space Administration (NNX14AT57A).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

EC	Enhancement constraint
LP	Linear programming
MILP	Mixed-integer linear program
OCPP	Overlap cuboid packing problem
SOLV	Spacecraft optimization layout and volume

CHAPTER 1: INTRODUCTION

Consider the problem of packing a set of items into a container, where items are cuboids with various sizes. The primary objective of this problem is to minimize the total volume of the container once all items have been packed. However, it is possible for this problem to be multi-objective by considering other criteria such as desired adjacency between items and packing costs. This problem is known as the packing problem [5].

The packing problem considered in this research stems from the effort to generate spacecraft interior layout designs. The container in this context will be referred to as the spacecraft module and the items to be packed will be referred to as gradient cuboids. A gradient cuboid represents a task volume where astronauts perform a task assigned to the volume. It is possible that there are multiple occurrences of the same gradient cuboid, e.g., multiple 'sleep' task volumes, to accommodate multiple astronauts. It is also important to astronauts that certain task volumes are placed next to one another for efficiency ,(i.e., kitchen and eating area), as well as certain task volumes being as far apart from each other (i.e., sleeping quarters and bathroom). In order to incorporate two performance criteria, the total volume of the module and adjacency preferences between task volumes, this study will propose a mathematical optimization model, called a mixed-integer linear program (MILP), where these two criteria are embodied in the objective function.

There are three layout design configurations that are considered for this study. The first case is where the dimensions of the module are unbounded in three-dimensions (which implies that no restriction on the size of the module is necessary). The second case is where the height and width dimensions of the module are limited (so-called

horizontal layout design configuration). The final case is where the floor dimensions of the module are limited (so-called vertical layout design configuration). These layout designs depend on the design requirements that set are set forth prior to executing the optimization.

Unlike conventional packing problems, a distinctive feature of the considered problem is to allow cuboids to overlap to a certain extent. This is because of the fact that it is not necessary to occupy the entire task volume exclusively when performing some tasks. Besides, some tasks can be performed at exclusively different times. In an initial base scenario, it is assumed that only side-to-side overlap between cuboids is allowed, while vertical overlap is prohibited. Overlap percentages between all pairs of cuboids are required before the optimization can be executed and are computed based on the probabilistic usage of the volume for a task [1]. Allowing overlap also presents the opportunity to further minimize the volume of the module since the gradient cuboids are allowed to share space. This problem variant can be identified as the overlap cuboid packing problem (OCPP).

It should be noted that the conventional cuboid packing problem is NP-hard. Adding overlap will create extra complexity. Therefore, one big issue with finding an optimal solution to this problem is the computational effort that is required to find an optimal solution. Although the problem of interest may not sound difficult for only a few items, (in which many feasible solutions still exist), it can become much more complicated very quickly as the number of items to pack grows. There are numerous heuristic methods that can provide solutions in a reasonable amount of time. However, heuristic methods fail to guarantee an optimal solution, but are preferred by many

decision makers due to their computational efficiency. Nonetheless it is desired to find an optimal solution via exact methods and this research aims at not only proposing exact formulations for the overlap packing problem but also alleviating computational burden to help improve the convergence rate of the solution method. This thesis is organized as follows.

In Chapter 2, an overview of existing literature within the problem domain of packing optimization will be discussed. In Chapter 3, the MILP will be formulated in the base case where the dimensions of the module are unbounded. Also, additional constraints will be introduced that allow for the other layout designs mentioned earlier to be constructed. In Chapter 4, a computational study will be presented, including analysis on the base model efficiency for small-scale and large-scale problems by using randomly generated cuboids, as well as comparing various layout design alternatives based on the weighted preference between adjacency and volume. In Chapter 5, model enhancements and reformulations to the base model will be introduced that are intended to reduce the computational efforts. In Chapter 6, an additional computational study will be presented, including a comparison between the MILP without the time-saving constraints and the MILP with the time-saving constraints. In Chapter 7, a conclusion to this research will be presented, as well as to discuss further extensions to this research.

CHAPTER 2: LITERATURE REVIEW

Extensive research has been performed in the field of packing problems with the primary objective of minimizing the volume of a container that holds a set of items. Cuboid packing problems can be applied to various industries due to their broad applicability, primarily in manufacturing and distribution, to assist in minimizing logistics costs [2]. While packing cuboids in a minimally sized container is the goal of packing problems there can be additional requirements depending on the application of the problem. Examples of such requirements include ensuring vertical stability such that items are placed on the base of the container or above other items, arranging items such that edges of each item are parallel or perpendicular to the container, and enabling rotation, which is dependent on the type of objects that are packed. Various problems may require additional or fewer constraints depending on its context. Efforts have been made to categorize such various types of packing problems to assist in determining which problem type is most applicable for a certain situation. As a result of these efforts, a typology was developed by a group of researchers.

According to [5], packing problems can be modeled under two distinct frameworks both with unique objectives. The first framework is known as output maximization, which follows the concept of maximizing the assignment of a set of items to a large container of limited size (i.e., pack as many items into a container as possible). The second framework is known as input minimization, which follows the concept of assigning a set of items while minimizing the dimensions of the large container (i.e., pack all items of interest into a container such that the size of the container is minimized).

Although both frameworks are slightly different from one another the underlying concept is similar which is packing a set of items into a container.

In general, there are three cases to describe the relationship between items that are to be packed. First, all items are identically sized to one another so that they possess the same dimensions (identical items). Second, items can be categorized such that items having same dimensions are grouped together in the same category while items from different categories have different sizes (weakly heterogenous assortment). Third, items are all treated as individual from one another even if there exist a small number of items that are identical to one another (strongly heterogenous assortment). For the third case, it makes sense to treat all items as unique from another since it may not be worth the effort to categorize the items if only a few of them are identical to one another. To expand upon the nature of items to be packed there is no restriction as to what the shape of the items can be. In the two-dimensional case, items can be rectangular, circular, etc. and in the three-dimensional case items can be rectangular prisms, spherical, etc. It is also possible for items to have irregular shapes [5].

Within the problem domain of packing optimization, there exist several problem types that can be applied depending on the context of a given problem. According to [5], these problem types include the identical item packing problem, placement problem, knapsack problem, open dimension problem, cutting stock problem, and the bin packing problem (where the first three problem types are classified as output maximization problems, and the rest are classified as input minimization problems). The identical item packing problem involves packing a set of identical items into a container. The placement problem involves packing a set of items belonging to multiple categories, where

categories are formed such that identical items are in the same category, into a container. The knapsack problem involves packing a set of items into a container with a fixed size. Assuming that all items cannot be packed into the container only a subset can be placed into the container such that the total value of packed items is maximized. The open dimension problem involves packing a set of items into a container that can have a variable size in one or more dimensions. In this type of problem, it is desired to fit all items into the container while changing one or more dimensions of the container such that its total volume is minimized. In the context of the packing problem, the cutting stock problem can be interpreted as packing multiples of assorted items into a set of containers with minimally wasted space. The bin packing problem is a special case of the cutting stock where assorted items are packed in a way that the total number of identically-sized containers is minimized.

In complexity theory, there are two distinct classes that are used to classify problems, P and NP [31]. Members of class P can be solved in polynomial time, whereas it has not been determined whether members of class NP are solvable in polynomial time of the problem size [31]. There are certain problems belonging to NP that by default require all similar problems to belong to the same class. Problems that have this characteristic are known as NP-complete [31]. If it can be proven that a problem belonging to NP requires exponential time to solve, an NP-complete problem will require it as well [31]. Similarly, the complexity of algorithms can be classified by their solution times and/or storage requirements. Exponential time algorithms consume exponential time of the problem size and require for a large solution space of exponential size if searched exhaustively, which is referred to as a brute-force search [31]. Polynomial time

algorithms are more useful due to their efficiency, whereas exponential time algorithms are rarely useful due to how computationally demanding they are [31]. When proving that an algorithm belongs to class P, it is necessary to provide its performance in the worst-case on a specific input length, and then analyze each step in the algorithm in order to verify that it can be implemented in polynomial time on a deterministic model [31]. Packing problems in the two-dimensional case have been deemed to be NP-complete [3], as well as in the three-dimensional case where the height of the container is unbounded [4]. As a result, heuristic methods are more predominant for the packing problem in order to quickly find a solution.

For many packing optimization problems, various algorithms and heuristic methods are used to assist in finding a good solution in a reasonable amount of time. In [11], the largest area first-fit algorithm was formulated in which a set of rectangular prisms are packed into a container where its height is unbounded. The item with the highest surface is placed first so the width and length dimensions of the container can be determined, and then the remaining items are placed accordingly so the height of the container is minimized. In [12], a tree-search algorithm was formulated which allows for a set of rectangular prisms to be packed. Computational complexity is reduced using this algorithm by decomposing the problem into a set of lower dimensional problems, when appropriate, until a good solution is found. In [13], a simulated annealing algorithm was formulated that allows for a solution to be found after a finite number of iterations have been performed, or if an acceptable wasted space threshold is satisfied for a set of objects. However, by the nature of heuristics, these methods fail to guarantee that an optimal solution to the problem can be found. Within these algorithms, various solution

strategies have been studied including, but not limited to, layering, cutting and residual space strategies.

The layering strategy requires the container to be separated into a set of layers and in order to progress to a new layer existing layers must be packed to the maximum capacity. This allows for items to be placed along the base of the container first, and then other items are placed above [6]. In [7], a layering strategy was utilized in solving a one-dimensional packing problem where the width and length of the container are fixed and is composed of two-dimensional packing problems where items are packed such that the usage of the surface area for a layer is best utilized. The layering strategy is also known as a block-building strategy, where the width of the container is filled to maximum capacity first, then its length and finally its height [19]. In [22], an algorithm was formulated that packs a set of items, called blocks, inside of a container during each iteration using a best-fit heuristic while ensuring that a reference length threshold is satisfied. The reference length is measured for each iteration, like the layering strategy, until all items are packed into the container.

The cutting strategy requires the container to be cut into horizontal and vertical strips. Once these strips are made, items are packed into them with the final representation of the container being classified as a pattern [18]. In [21], a single dimension open dimension problem was considered where the container is cut into horizontal and vertical strips. Items are placed within these generated strips until a solution is found.

The residual space strategy allows for the generation of small-sized containers to be generated inside of the large container each time an additional item is packed.

Typically when using this strategy, it is required for additional items to be packed along one of the corner points for each residual space in hopes of minimizing the number of residual spaces that are created [16]. The residual space strategy is also known as the bottom-left with fill approach, where it is desired to place the first item at the bottom-left hand corner of the container and place additional items beside one another until the container is packed [20].

It is possible for an algorithm to utilize several solution strategies throughout its entire procedure and are typically known as hybrid algorithms. Hybrid algorithms consist of multiple stages that need to be solved before a solution can be given and use multiple analytical tools. A majority of algorithms that are implemented for solving packing problems are hybrid since further improvement in computational efficiency is common. In [18], a hybrid heuristic for the two-dimensional bin packing problem was formulated where the first-stage consists of cutting slips from the container that items can be packed in, thus creating a pattern, using a linear programming model, and the second-stage requires an integer program to be solved for minimizing the number of patterns, or bins, that are used. In [23], the block-building strategy was used for the single container loading problem. Initially, the empty container is the only space that is available for items to be packed in. After the first item is packed, new residual spaces are created until no remaining residual spaces exist.

Many researchers have also utilized genetic algorithms for solving the packing problem. Genetic algorithms require a particular problem to be solved multiple times in order to find the best possible solution within a population of solutions [25]. They typically start with a randomly generated set of chromosomes that represent solutions and

focus on the ones that are more likely to reproduce better solutions [25]. Once this population of chromosomes evolve via genetic operations many solutions can exist and are compared with one another in selecting the best solution [25]. In [6], a hybrid genetic algorithm was formulated such that a set of rectangular items need to be packaged in a large container where its space is used most optimally using a layering strategy. For the case where one item is above another there must be no lateral overhang meaning that there should be no empty space surrounding these two items. This helps in using space most efficiently since it would be computationally difficult to pack another item in a small empty space. In [19], a genetic algorithm was applied along with bottom-left with fill and block-building strategies, as well as a greedy heuristic algorithm. The greedy heuristic allows for the spare space in the container to be minimized and for the weight to be distributed such that the container remains balanced.

It is also possible to formulate packing optimization models into a constrained mathematical program such that an optimal solution to the problem is guaranteed. In [8], an integer program was formulated for the strip packing problem in two dimensions. In [15], a mathematical linear model was formulated which allows for various constraints to be considered including container weight limit, load stability, weight distribution and the fragility of items, while minimizing the unused volume of the container. Depending on the formulation of the model it is possible for the container to be rectangular or a truncated parallelepiped. In [2], a highly non-linear and non-convex mixed-integer program was formulated where the objective function is to minimize the volume of an unbounded container. Since this problem is non-linear it is difficult to find an optimal solution to the problem even when existing numerical approximation techniques and

algorithms are used. As a result, a reformulated objective function was suggested that reduces it to only a single variable, resulting in a quadratic objective function [2]. The packing of circular and spherical items is also a popular research area. In literature, the container can be considered in two- and three-dimensions, where the container can have a square, rectangular, or circular region [26]. A nonlinear programming model was formulated to solve this problem in two-dimensions using an energy function, such that it is minimized subject to items being packed inside of the square without overlap [27]. In [28, 29], researchers were able to formulate non-linear programming models in three-dimensions where the container has an unbounded height and can take the shape of a parallelepiped and a right circular cylinder where identical spheres are packed. In [30], a similar model was proposed for non-identical spheres being packed into a parallelepiped. Similar to packing rectangular shaped items, heuristic algorithms are more predominant as a solution tool for solving this problem variant.

There exist algorithms and mathematical models that can assist in finding a solution to items of irregular shape, where it is not possible to classify them with any known geometrical shape. Irregular shapes can be represented as convex or non-convex polygons, with the latter situation making the problem more difficult. Within [17], two mixed-integer linear programming models were formulated that provide a solution to the strip packing problem where items of irregular shape can be considered. For each irregularly shaped item they can be broken up into a set of convex polygons. In [24], irregular shaped items can be packed by bounding them in an arbitrary cuboid such that the entire item is placed inside and the dimensions of the cuboid are minimized. Orientation of the items are considered as a result and are loaded into the container using

bottom-left with fill and find-face algorithms. In [20], a hybrid algorithm is presented that is composed of two methods. The first method uses the bottom-left with fill strategy and uses simple geometric tools to speed up the computational result. The second method consists of a genetic algorithm where the best solution can be found as its population expands

Due to the computational complexity of packing problems it is desired to reduce the computational burden that is experienced when solving the problem. Noting that packing problems are typically formulated as MILPs, one mathematical modeling technique of interest is to add valid inequalities in order to tighten the linear programming (LP) relaxation of the MILP. In general, an inequality is said to be valid if it is satisfied by all other possible solutions that belong to the feasible region [9]. The benefit of implementing valid inequalities into a mathematical model allows for the feasible region to shrink in size, or tighten, while ensuring that the original problems constraints are still enforced. Tightening the feasible region may lead to reduced computational times. It is possible to formulate valid inequalities by understanding the problem of interest and the structure of the model. However, there exists more mathematically rigorous techniques for generating constraints of this type. In [10], a general procedure is proposed for generating valid inequalities for various integer programs.

As a result of reviewing relevant literature, it was noticed that there is a common objective of minimizing container volume, which is similar to maximizing the utilization of the container. The minimization of container volume can be accomplished by fixing two-dimensions of the container and minimizing the free dimension, or by solving a

highly non-convex problem that requires heavy computational effort. Furthermore, none of the reviewed packing literature considers overlapping between items to be packed. Also, adjacency requirements for items do not seem to be prevalent in literature which results in packing layouts where things appear to be randomly placed. In this research, however, it is desired to allow overlap between gradient cuboids in order to further minimize the dimensions of the spacecraft by allowing certain task volumes to share space. It is also desired to enforce adjacency requirements between task volumes based on the decision makers' preference, which will ultimately result in packing (i.e., layout designs) that are more efficient in volume utilization and are astronaut friendly.

CHAPTER 3: PROBLEM DESCRIPTION

In this chapter, an MILP is proposed in order to generate an arrangement of gradient cuboids that are to be packed inside of the module. To formulate this problem under the general setting, consider *N* gradient cuboids that need to be packed in the module. The shape of the gradient cuboids and the module are rectangular where each cuboid $i = 1, \dots, N$ is characterized by its *length* (l_i), *width* (w_i), and *height* (h_i), with designated orientation as illustrated in Figure 1.

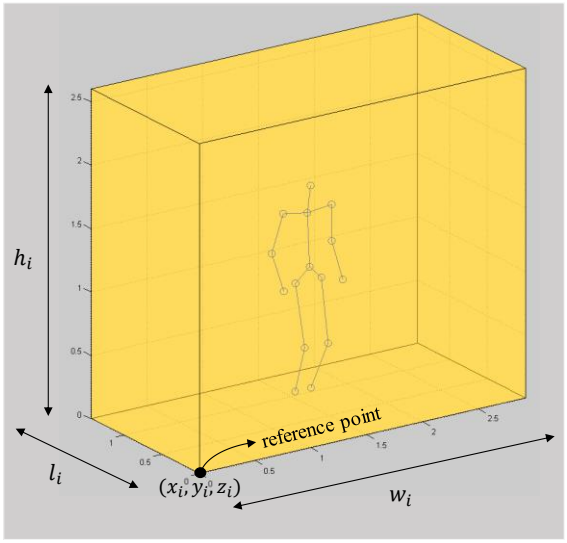


Figure 1: Example of gradient cuboid

Recall that gradient cuboids correspond to the various types of tasks. Hence, the size of a cuboid depends on the volume required to perform the task. Certain types of tasks can be performed simultaneously and multiple identically-size cuboids are needed for such tasks.

The performance criteria for the OCPP are the total volume of the container and the extent of adjacency between gradient cuboids. It is obvious that the volume of the module is the product of the total distance along the x-, y-, and z-axis. As mentioned

earlier when referencing [2], including the actual volume of the module results in a highly non-linear model that is hard to solve. In order to overcome this issue, it is proposed to include the estimated volume of the module in the objective function. The estimated volume can be formally stated as the sum of all distances across the x-, y-, and z-axis. Including this sum in the objective function will result in the MILP to attempt to minimize each dimension of the module, which is similar to minimizing its volume. The total distance between cuboid *i* and cuboid *j* is measured as the Manhattan distance, or the l_1 norm between their respective center points. In order to enforce that the total distance between cuboid *i* and cuboid *j* is minimized, penalties will be assigned to the decision variables in the objective function that are associated with all pairs where index *i* is less than *j*. However, it is not necessary for every pair of gradient cuboids to be adjacent to one another. A higher penalty indicates that it is more important for cuboid *i* and cuboid *j* to be adjacent, and a lower penalty indicates that it is less important.

Note that having two objectives results in a bi-objective optimization model. A widely practiced approach for a bi-objective optimization model is scalarization. In this study, a weight factor, $\alpha \in (0,1)$, is applied to the adjacency measure, where a higher weight implies that a relatively higher priority is given to minimizing the adjacency measure.

Several assumptions will now be stated that influence the structure of the MILP. Within this model, it is possible for gradient cuboids to be rotated 90 degrees along the x-y plane (Assumption 1). Only rotation along the x-y plane is permitted in order to guarantee that the gradient cuboid is always standing upright. This assumption addresses possible gravity that is generated within the module. Even without gravity, having a fixed

vertical orientation may help astronauts navigate as they are used to it on Earth. Also, it is only possible for overlap between gradient cuboids to occur along the x- and y-axis (Assumption 2). Some gradient cuboids may require various types of equipment to be placed along its base, so it is necessary to guarantee that the base of the gradient cuboid will not share space with any other gradient cuboids. Next, there is a special type of gradient cuboid that corresponds to the ingress/egress task, and hence, must be connected to the wall of the module (Assumption 3). This assumption is intuitive since the only way for astronauts to enter/exit the module is through one of its walls.

The general base model will be first formulated where the dimensions of the module are unbounded. Toward the end of the chapter, additional constraints will be introduced to the base model so that additional layout design alternatives can be constructed. Before the general model can be introduced, it is necessary to formally state the notation that will be used for various parameters and decision variables that are included in the MILP. The following notations will be used to denote problem parameters and variables:

(w_i, l_i, h_i): parameters representing the width, length, and height of cuboid i.

$M \gg 1$: parameter representing a large constant.

o_{ij}: parameter representing allowed overlap percentage between cuboid i and cuboid j. This parameter is not orientation specific, meaning that overlap between cuboids i and j is applied in any direction.

$\alpha \in (0, 1)$: parameter representing the weighting factor for the adjacency measurements. Accordingly, $(1 - \alpha)$ represents the weighting factor for the size of the container.

A_{ij}: parameter representing penalty proportional to the distance between cuboids i and j. This parameter will address adjacency requirements between cuboids i and j, where a larger value of A_{ij} implies a proximity between cuboids i and j is desired.

(X, Y, Z): continuous variables representing the dimensions of the cuboid container.

(x_i, y_i, z_i): continuous variables representing the coordinates of the reference point of cuboid i in three-dimensional Cartesian coordinate system (x, y, z) (see Figure 1 for the reference point of an example cuboid).

The following binary variables represent relative location of cuboid *i* with respect to cuboid *j*, while not infringing overlap allowance:

$p_{ij} = 1$ if cuboid i is to the left of j; otherwise $p_{ij} = 0$.

$q_{ij} = 1$ if cuboid i is in front of j; otherwise $q_{ij} = 0$.

$r_{ij} = 1$ if cuboid i is above j; otherwise $r_{ij} = 0$.

The following binary variables represent the orientation of cuboid *i* with respect to the x-axis:

$I_i^w = 1$ if cuboid i's width is parallel to x-axis; otherwise $I_i^w = 0$.

The following binary variables represent the relative orientation of cuboid *i* and cuboid *j* with respect to the x-axis:

$I_{ij}^{fg} = 1$ if cuboid i's width or length, i.e., $(f \in \{w, l\})$, is parallel to the x-axis and cuboid j's width or length, i.e., $g \in \{w, l\}$, is parallel to the x-axis, otherwise $I_{ij}^{fg} = 0$. The interpretation of these variables is demonstrated in Figure 2.

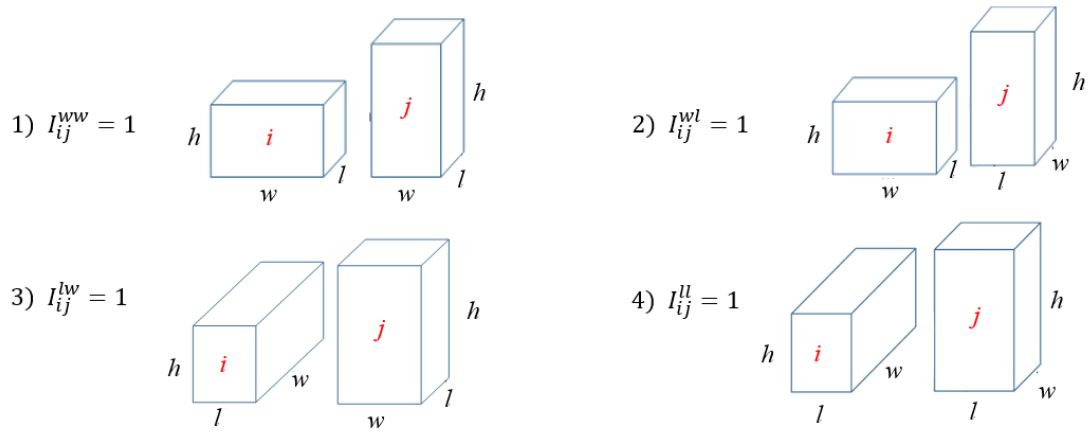


Figure 2: Binary variables representing orientaion of cuboids *i* and *j* with respect to the x-axis. 1) When the binary variable, I_{ij}^{ww}, is equal to 1, the widths of both cuboids i and j are parallel to the x-axis. 2) When the binary variable, I_{ij}^{wl}, is equal to 1, the width of cuboid *i* and the length of cuboid *j* are parallel to the x-axis. 3) When the binary variable, I_{ij}^{lw}, is equal to 1, the length of cuboid *i* and the width of cuboid *j* are parallel to the x-axis. 4) When the binary variable, I_{ij}^{ll}, is equal to 1, the lengths of cuboids *i* and *j* are parallel to the x-axis.

The following binary variables determine the wall of the container to which the hatch cuboid is attached:

$I_H^x = 1$ if the hatch cuboid is attached to the y-z plane; otherwise $I_H^x = 0$.

$I_H^y = 1$ if the hatch cuboid is attached to the x-z plane; otherwise $I_H^y = 0$.

$I_H^z = 1$ if the hatch cuboid is attached to the x-y plane; otherwise $I_H^z = 0$.

(c_i^x, c_i^y, c_i^z): continuous variables representing the coordinates of the center point of cuboid i.

d_{ij}: continuous variables indicating the distance between cuboids *i* and *j* measured as $\|(c_i^x, c_i^y, c_i^z) - (c_j^x, c_j^y, c_j^z)\|_1$.

Using the notation defined above, the proposed mixed-integer program (MILP) of the base model of the OCPP can be presented as follows:

Minimize $\alpha \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} A_{ij} d_{ij} + (1-\alpha)(X + Y + Z)$ (1)

subject to

$$x_j \geq x_i - M(1-p_{ij}) + \left(w_i - \min\{.01 o_{ij} w_i, .01\, o_{ji} w_j\}\right) I_{ij}^{ww} + \left(w_i - \min\{.01 o_{ij} w_i, .01 o_{ji} l_j\}\right) I_{ij}^{wl} + \left(l_i - \min\{.01 o_{ij} l_i, .01 o_{ji} w_j\}\right) I_{ij}^{lw} + \left(l_i - \min\{.01 o_{ij} l_i, .01 o_{ji} l_j\}\right) I_{ij}^{ll} \quad \forall\, i,j \quad i \neq j \quad (2)$$

$$y_j \geq y_i - M\left(1-q_{ij}\right) + \left(l_i - \min\{.01 o_{ij} l_i, .01 o_{ji} l_j\}\right) I_{ij}^{ww} + \left(l_i - \min\{.01 o_{ij} l_i, .01 o_{ji} w_j\}\right) I_{ij}^{wl} + \left(w_i - \min\{.01 o_{ij} w_i, .01\, o_{ji} l_j\}\right) I_{ij}^{lw} + \left(w_i - \min\left(\{.01 o_{ij} w_i, .01\, o_{ji} w_j\}\right)\right) I_{ij}^{ll} \quad \forall\, i,j \quad i \neq j \quad (3)$$

$$z_j \geq z_i + h_i - M(1-r_{ij}) \quad \forall\, i,j \quad i \neq j \quad (4)$$

$$p_{ij} + p_{ji} + q_{ij} + q_{ji} + r_{ij} + r_{ji} \geq 1 \quad \forall\, i,j \quad i < j \quad (5)$$

$$I_{ij}^{ww} \leq I_i^w \quad \forall\, i,j \quad i \neq j \quad (6)$$

$$I_{ij}^{ww} \leq I_j^w \quad \forall\, i,j \quad i \neq j \quad (7)$$

$$I_{ij}^{ww} \geq I_i^w + I_j^w - 1 \quad \forall\, i,j \quad i \neq j \quad (8)$$

$$I_{ij}^{wl} \leq I_i^w \quad \forall\, i,j \quad i \neq j \quad (9)$$

$$I_{ij}^{wl} \leq 1 - I_j^w \quad \forall\, i,j \quad i \neq j \quad (10)$$

$$I_{ij}^{wl} \geq I_i^w + \left(1 - I_j^w\right) - 1 \quad \forall\, i,j \quad i \neq j \quad (11)$$

$$I_{ij}^{lw} \leq 1 - I_i^w \quad \forall\, i,j \quad i \neq j \quad (12)$$

$$I_{ij}^{lw} \leq I_j^w \quad \forall\, i,j \quad i \neq j \quad (13)$$

$$I_{ij}^{lw} \geq (1 - I_i^w) + I_j^w - 1 \quad \forall\, i,j \quad i \neq j \quad (14)$$

$$I_{ij}^{ll} \leq 1 - I_i^w \quad \forall\, i,j \quad i \neq j \quad (15)$$

$$I_{ij}^{ll} \leq 1 - I_j^w \quad \forall\, i,j \quad i \neq j \quad (16)$$

$$I_{ij}^{ll} \geq (1 - I_i^w) + (1 - I_j^w) + 1 \qquad \forall\, i, j \quad i \neq j \tag{17}$$

$$X \geq x_i + l_i + (w_i - l_i) I_i^w \qquad \forall\, i \tag{18}$$

$$Y \geq y_i + w_i + (l_i - w_i) I_i^w \qquad \forall\, i \tag{19}$$

$$Z \geq z_i + h_i \qquad \forall\, i \tag{20}$$

$x_i \leq M(1 - I_H^x)$ where i represents the index of the hatch cuboid (21)

$y_i \leq M(1 - I_H^y)$ where i represents the index of the hatch cuboid (22)

$z_i \leq M(1 - I_H^z)$ where i represents the index of the hatch cuboid (23)

$$I_H^x + I_H^y + I_H^z = 1 \tag{24}$$

$$c_i^x = x_i + \frac{w_i}{2} * I_i^w + \frac{l_i}{2} * (1 - I_i^w) \qquad \forall\, i \tag{25}$$

$$c_i^y = y_i + \frac{l_i}{2} * I_i^w + \frac{w_i}{2} * (1 - I_i^w) \qquad \forall\, i \tag{26}$$

$$c_i^z = z_i + \frac{h_i}{2} \qquad \forall\, i \tag{27}$$

$$d_{ij} \geq c_i^x - c_j^x + c_i^y - c_j^y + c_i^z - c_j^z \qquad \forall\, i, j \quad i < j \tag{28}$$

$$d_{ij} \geq -(c_i^x - c_j^x) + c_i^y - c_j^y + c_i^z - c_j^z \qquad \forall\, i, j \quad i < j \tag{29}$$

$$d_{ij} \geq c_i^x - c_j^x - (c_i^y - c_j^y) + c_i^z - c_j^z \qquad \forall\, i, j \quad i < j \tag{30}$$

$$d_{ij} \geq c_i^x - c_j^x + c_i^y - c_j^y - (c_i^z - c_j^z) \qquad \forall\, i, j \quad i < j \tag{31}$$

$$d_{ij} \geq -\left(c_i^x - c_j^x\right) - (c_i^y - c_j^y) + c_i^z - c_j^z \qquad \forall\, i, j \quad i < j \tag{32}$$

$$d_{ij} \geq -(c_i^x - c_j^x) + c_i^y - c_j^y - (c_i^z - c_j^z) \qquad \forall\, i, j \quad i < j \tag{33}$$

$$d_{ij} \geq c_i^x - c_j^x - (c_i^y - c_j^y) - (c_i^z - c_j^z) \qquad \forall\, i, j \quad i < j \tag{34}$$

$$d_{ij} \geq -\left(c_i^x - c_j^x\right) - (c_i^y - c_j^y) - (c_i^z - c_j^z) \qquad \forall\, i, j \quad i < j \tag{35}$$

$$x_i, y_i, z_i \geq 0 \qquad \forall\, i. \qquad (36)$$

As mentioned earlier, the objective function of the MILP consists of two performance measures: the size of the container and the adjacency requirements. The l_1 norm of (x, y, z) is used to measure the size of the container, while the weighted sum of distances between each pair of cuboids is used to measure the adjacency requirements. The latter is designed to penalize the objective function for having large distances between cuboids that are required to have proximity. As mentioned earlier, the penalty A_{ij} can be prescribed based on the proximity requirements.

Constraints (2) and (3) allow the cuboids to overlap each other up to the allowable overlap percentage between cuboid i and j on the x- and y-axis. These conditions indicate that if cuboid j is to the right of, behind of, or above cuboid i, then the reference point of cuboid j must be greater than or equal to the reference point of cuboid i plus the width, length, or height of cuboid j, respectively. These constraints also ensure that the orientation of two cuboids are correctly reflected on the x-y plane. Constraint (4) ensures no cuboids overlap vertically. Constraint (5) ensures that cuboid i must be placed to the left of, to the right of, in front of, behind, below, or above cuboid j to guarantee that these two cuboids are not overlaid more than allowed percentages.

Constraints (6) – (17) represent the relationship between I_{ij}^{fg} and I_i^w for $f, g \in \{w, l\}$. They can be interpreted as linearization constraints for the following nonlinear relationship:

$$I_{ij}^{ww} = I_i^w I_j^w$$

$$I_{ij}^{wl} = I_i^w \left(1 - I_j^w\right)$$

$$I_{ij}^{lw} = (1 - I_i^w)I_j^w$$

$$I_{ij}^{ll} = (1 - I_i^w)(1 - I_j^w)$$

Constraints (18) – (20) ensure that all cuboids are packed inside the container by making the width, length, and height of the container larger than or equal to the reference point of each cuboid plus its width, length, and height, respectively. Constraints (21) – (24) guarantee that the hatch cuboid must be attached to a wall within the container.

The center points for all cuboids are computed and used to measure the distance between cuboids *i* and *j*. Constraints (28) – (35) enforce lower bounds on the distance between cuboid i and cuboid j. Together with minimizing the objective function, d_{ij} results in the l_1-distance between center points of two cuboids. Constraint (36) implies that the coordinates of the cuboids are non-negative. In case that overlap amongst cuboids along the z-axis is allowed, constraint (4) can replaced with (37).

$$z_j \geq z_i + h_i - M(1 - r_{ij}) - \min\left(.01o_{ij}h_i, .01o_{ji}h_j\right) \quad \forall\, i,j \qquad i \neq j \tag{37}$$

If it is desired to further limit any dimension of the container due to design requirements, additional constraints can be added. In particular, we consider two types of design requirements, horizontal and vertical. Horizontal layout design requires that the width and height of the container are limited by a prescribed value δ. On the other hand, in the vertical design, the width and the length of the container are limited by a threshold value δ. For the horizontal layout design case, constraints (38) and (40) are necessary. For the vertical layout design case, constraint (38) and (39) are necessary.

$$X \leq \delta \tag{38}$$

$$Y \leq \delta \tag{39}$$

$$Z \leq \delta \tag{40}$$

The size of the proposed MILP model can be specified by the number of binary and continuous variables as well as the number of constraints. First, the number of inequality constraints is given by

$$f_{ineq}(N) = 3 + 3 + 3N + \ 3N + 12N(N-1) + 4N(N-1) + 3N(N-1) + \frac{N(N-1)}{2}$$

which is equivalent to:

$$f_{ineq}(N) = \frac{39N^2 - 27N + 12}{2}.$$

From (24) – (27), the number of equality constraints included in the base model is $f_{eq}(N) = 3N + 1$. The number of binary variables included in the base model can be calculated by

$$f_{binary}(N) = 3 + N + 4N(N-1) + 3N(N-1)$$

which is equivalent to:

$$f_{binary}(N) = 7N^2 - 6N + 3.$$

The number of continuous variables included in the base model is measured by

$$f_{cont}(N) = 3 + 3N + 3N + \frac{N(N-1)}{2}$$

which is equivalent to:

$$f_{cont}(N) = \frac{N^2 + 11N + 6}{2}.$$

Taking the sum of f_{ineq} and f_{eq} will yield the total number of constraints and taking the sum of f_{binary} and f_{cont} will yield the total number of variables in the base model. Now that the formulation to the MILP has been presented, Chapter 4 will consist of a numerical study that was performed in order to demonstrate the model's ability to produce layout designs based on various design requirements.

CHAPTER 4: COMPUTATIONAL STUDY

In this chapter, a computational study will be presented where the computational efficacy of the proposed model was analyzed on a set of test problems consisting of small- and large-scale problems. The small-scale problems were used to demonstrate how the MILP model effectively generates layout designs under different design assumptions, including different values of the weighting factor, α. On the other hand, the large-scale problem considers a realistic scenario where all necessary gradient cuboids are included. The small-scale problem will be presented first and then the large-scale problem will follow afterwards.

4.1 Small-Scale Computational Study

In order to demonstrate how the MILP model generates a layout design, the proposed design model was implemented on a test problem that consists of seven gradient cuboids including one exercise, one hygiene, one waste collection and management, three sleep, and one hatch cuboids. The dimensions of these five types of task volumes were provided by the gradient cuboid algorithm of SOLV [1] and are displayed in Table 1. Maximum pairwise overlap percentages between task volumes are presented in Table 2. In order to incorporate adjacency requirements for the MILP, several categories were used to describe the severity of adjacency between a pair of gradient cuboids. The categories for adjacency requirements and their respective penalty values used in the numerical example are displayed in Table 3. The assignment of adjacency requirements for each pair of task volumes is displayed in Table 4.

Table 1: Dimension of gradient cuboids – Five task volumes

Task Volume	Width	Length	Height
Exercise	2.92	1.43	2.60
Hygiene	1.42	1.99	2.46
Waste Collection and Management	3.37	1.35	2.49
Sleep	1.16	1.23	2.70
Hatch	2.69	1.54	1.95

Table 2: Maximum pairwise overlap percentage (%)

	Exercise	Hygiene	Waste Collection and Management	Sleep	Hatch
Exercise	-	6.37	6.37	6.37	6.37
Hygiene	6.37	-	11.78	11.78	11.78
Waste Collection and Management	6.37	11.78	-	13.01	13.01
Slee	6.37	11.78	13.01	-	6.6
Hatch	6.37	11.78	13.01	6.6	-

Table 3: Adjacency requirement categories

Category Index	Description	Penalty Value (A_{ij})
3	Proximity Desired	50
2	Some Proximity Desired	10
1	Neutral	1
0	Separation Desired	0

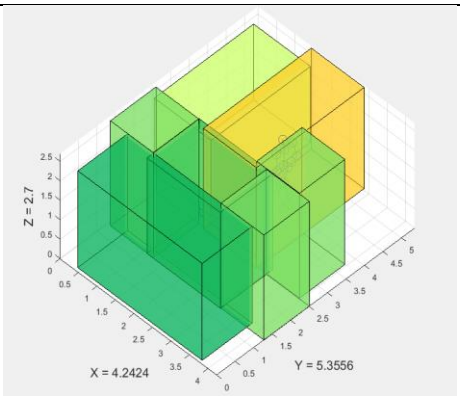
Table 4: Adjacency requirements

	Exercise	Hygiene	Waste Collection and Management	Sleep	Hatch
Exercise	1	10	1	0	1
Hygiene	10	1	50	10	1
Waste Collection and Management	1	50	1	1	1
Sleep	0	10	1	1	1
Hatch	1	1	1	1	1

To demonstrate how different design variations produce different layouts, this study ran the base design model, as well as both design variations of the single-dimensional packing problem (horizontal and vertical layout designs) with and without vertical overlap. For the single-dimensional packing optimization, the value of δ was selected as 4 and 5, resulting in two layouts for both layout designs. Weighting factor α in the objective function was set to 0.5 to obtain solutions from the models to be evaluated. In total, 10 layout designs were considered, as displayed in Table 5. The MILP models were created using Matlab [32] and solved by calling a commercial MILP solver, Gurobi Optimizer 7.5 [33]. The resulting 10 layouts are visually displayed in Figure 3.

Table 5: Layout design configurations

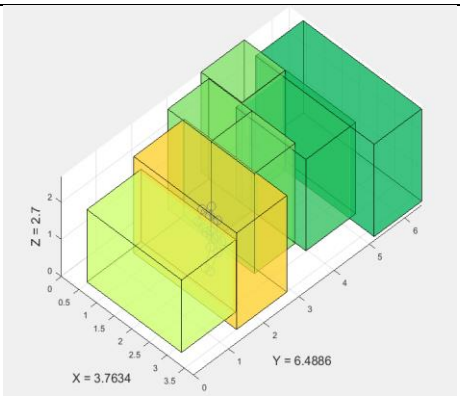
Design Name	Vertical overlap	Horizontal layout	Vertical layout	δ
overz_cuboid	x			
overz_horizontal_4	x	x		4
overz_horizontal_5	x	x		5
overz_vertical_4	x		x	4
overz_vertical_5	x		x	5
solidz_cuboid				
solidz_horizontal_4		x		4
solidz_horizontal_5		x		5
solidz_vertical_4			x	4
solidz_vertical_5			x	5



(a) overz_cuboid and solidz_cuboid

(X, Y, Z): (4.2424, 5.3556, 2.7)

Container volume: 61.3456

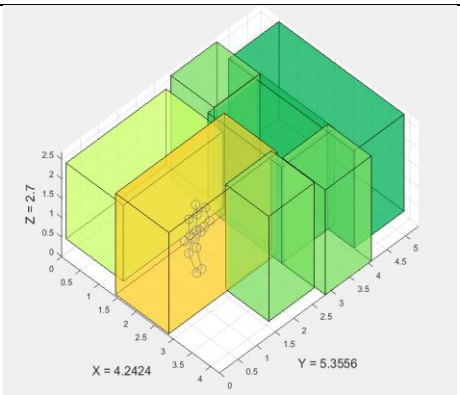


(b) overz_horizontal_4 and solidz_horizontal_4

(X, Y, Z): (3.7634, 6.4886, 2.7)

Container volume: 65.9318

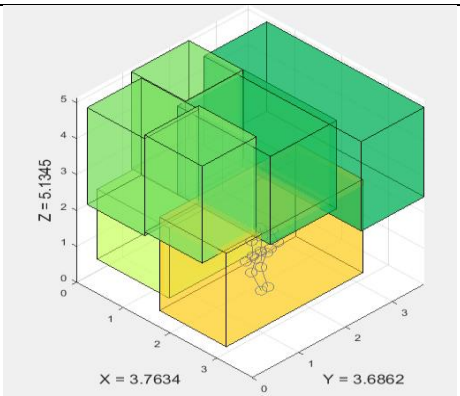
Figure 3: 10 layout designs for small-scale problem



(c) overz_horizontal_5 and solidz_horizontal_5

(X, Y, Z): (4.2424, 5.3556, 2.7)

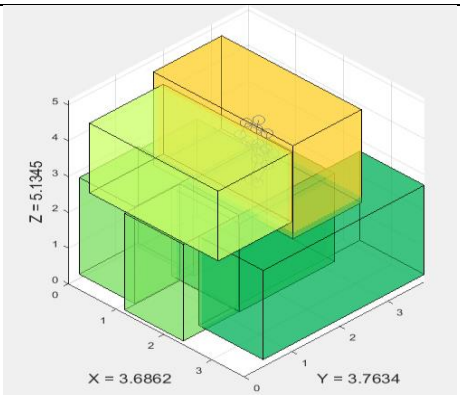
Container volume: 61.3456



(d) overz_vertical_4

(X, Y, Z): (3.7634, 3.6862, 5.1345)

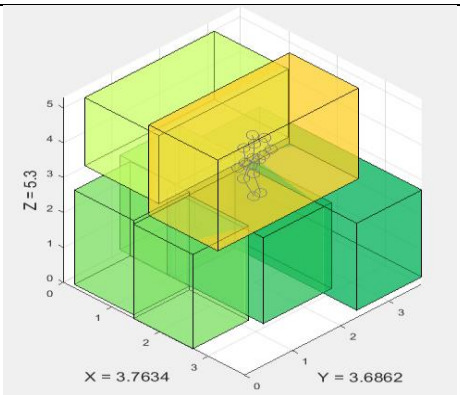
Container volume: 71.2291



(e) overz_vertical_5

(X, Y, Z): (3.6862, 3.7634, 5.1345)

Container volume: 71.2291

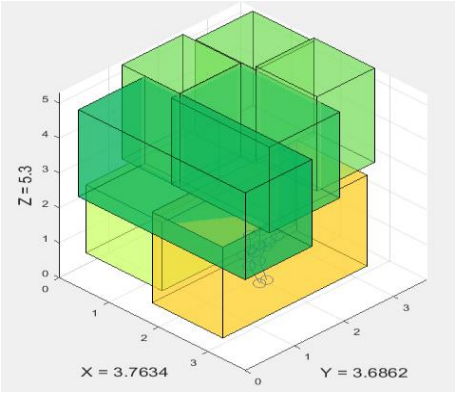


(f) solidz_vertical_4

(X, Y, Z): (3.7634, 3.6862, 5.3)

Container volume: 73.5250

Figure 3, continued



(g) solidz_vertical_5

(X, Y, Z): (3.7634, 3.6862, 5.3)

Container volume: 73.5250

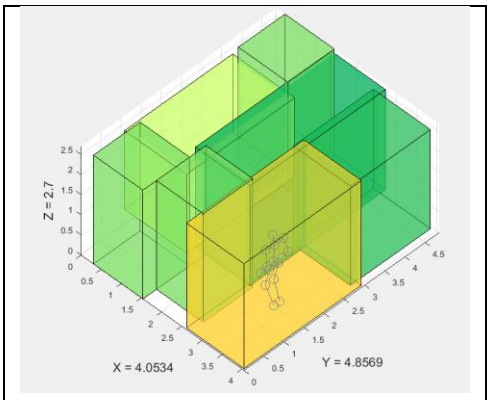
Figure 3, continued

To check computational times consumed to generate layout designs, a set of 19 scenarios were additionally created in which the dimensions of the gradient cuboids are randomly generated except for the hatch gradient cuboid. To be more specific, the dimensions of each cuboid were generated using uniform random variables, allowing 20% of variation from the original dimensions from Table 1. For example, a gradient cuboid representing 'Exercise' has the width of 2.92. In a new scenario, the width of the exercise task volume is generated from $[2.92 - 0.2(2.92), 2.92 + 0.2(2.92)]$. In total, 20 scenarios (one original and 19 randomly generated) were tested for each layout design. After all scenarios for a layout design variant are completed, solver times from each scenario were measured and averages of 20 scenarios are displayed in Table 6. Overall, the MILP was able to generate a layout design for seven cuboids within 5.01 to 1,055.76 seconds.

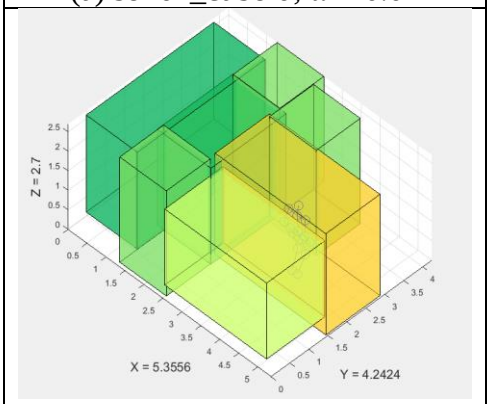
Table 6: Average solution time for scenarios

Layout Design	Average Solution Time (seconds)	Standard Deviation	Max.	Min.
overz_cuboid	245.25	225.71	791.64	52.84
overz_horizontal_4	16.27	12.07	56.26	5.30
overz_horizontal_5	98.59	103.59	405.38	13.25
overz_vertical_4	33.64	20.61	91.68	12.94
overz_vertical_5	74.10	81.20	315.99	25.39
solidz_cuboid	304.74	282.40	1055.76	45.72
solidz_horizontal_4	14.80	7.23	39.21	5.01
solidz_horizontal_5	76.35	98.86	431.28	9.24
solidz_vertical_4	41.78	76.97	365.81	11.53
solidz_vertical_5	68.94	61.83	265.21	23.11

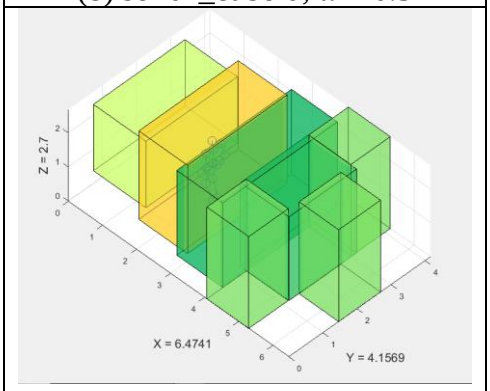
Recall that α was 0.5 for the small-scale problem that was presented earlier this chapter. To ascertain the effect of different values of α, the base design was tested with $\alpha \in [.01, .99]$ in increments of .01. For each value of α, two performance measures in the objective function are plotted to display α using a Pareto Frontier. The generated layouts with three values of α, 0.01, 0.05, 0.99, are displayed in Figure 4, and the Pareto Frontier is displayed in Figure 5. Observe that the cuboids are more densely packed in (a) of Figure 4, compared to the layout in (c) of Figure 4, because the size of the container is given a relatively higher priority in minimization.



(a) solidz_cuboid; α = 0.01



(b) solidz_cuboid; α = 0.5



(c) solidz_cuboid; α = 0.99

Figure 4: 10 layout designs for varying levels of α

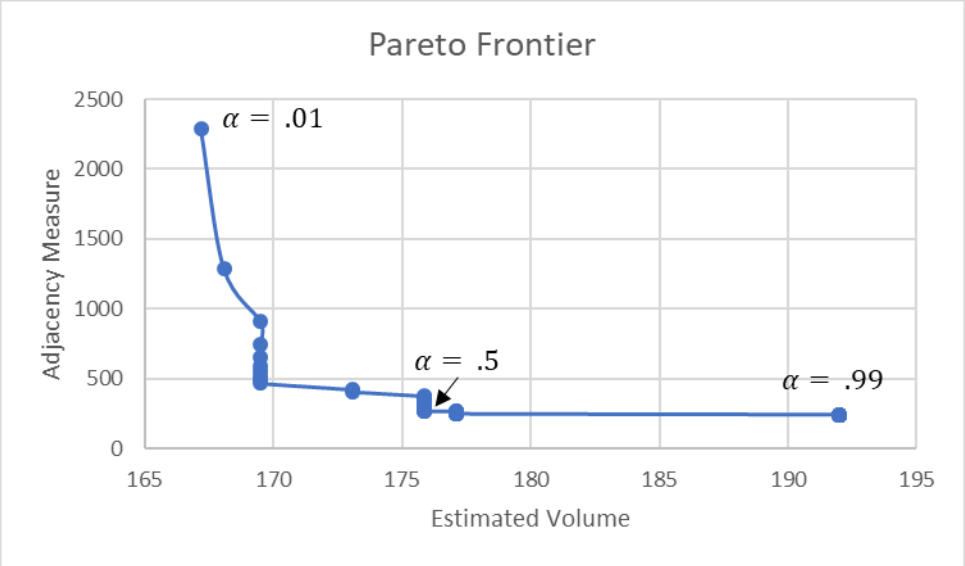


Figure 5: Pareto frontier

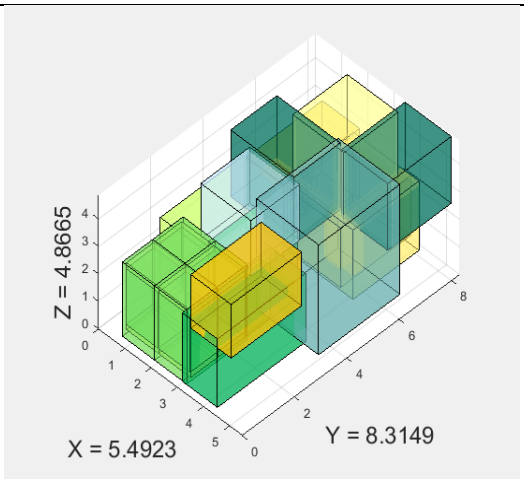
4.2 Large-Scale Computational Study

The large-scale problem consists of twenty-four gradients cuboids including one exercise, one hygiene, one waste collection and management, four sleep, one crew health and medical, four private personal activity, one food preparation, one group meet and eat, one recreation, one suit stowage and maintenance, four radiation shelter, one D&C console, two onboard research, and one hatch cuboids, which are associated with fourteen task volumes. The dimensions of these fourteen types of task volumes were provided by the gradient cuboid algorithm of SOLV [1] and are displayed in Table 7. Adjacency requirement categories in Table 3 were considered for the large-scale problem as well.

Table 7: Dimension of gradient cuboids – Twenty-four task volumes

Task Volume	Width	Length	Height
Exercise	2.92	1.43	2.60
Hygiene	1.42	1.99	2.46
Waste Collection and Management	3.37	1.35	2.49
Sleep	1.16	1.23	2.70
Crew Health and Medical	3.21	2.03	2.17
Private Personal Activity	1.06	1.23	2.43
Food Preparation	2.07	1.95	2.56
Group Meet and Eat	2.52	2.52	2.52
Recreation	2.70	2.13	2.16
Suit Stowage and Maintenance	2.37	3.11	4
Radiation Shelter	1.07	1.23	2.04
D&C Console	1.87	1.92	2.03
Onboard Research	2.43	1.77	2.43
Hatch	2.69	1.54	1.95

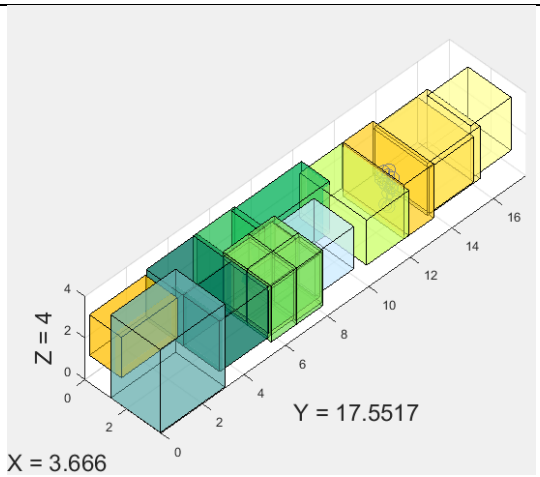
As done for the small-scale problem, we ran the base design model, as well as both design variations of the single-dimensional packing problem (horizontal and vertical layout designs) with and without vertical overlap. For the single-dimensional packing optimization, the value of δ was selected as 4 and 5, resulting in two layouts for both layout designs. Weighting factor α in the objective function was set to 0.5 to obtain solutions from the models to be evaluated. In total, 10 layout designs were considered, as displayed in Table 5. The MILP models were created using Matlab [32] and solved by Gurobi Optimizer 7.5 [33]. Considering the complexity of the problem that resulted in more than 1,000 seconds of solution time in a certain small-scale problem, the solution time limit was set to 4-hours. Solutions of all 10 problems were terminated by this 4-hour time limit and the optimality gaps were recorded. The resulting 10 layouts are visually displayed in Figure 6. The optimality gaps are displayed in Table 8.



(a) overz_cuboid

(X, Y, Z): (5.4923, 4.8665, 8.3149)

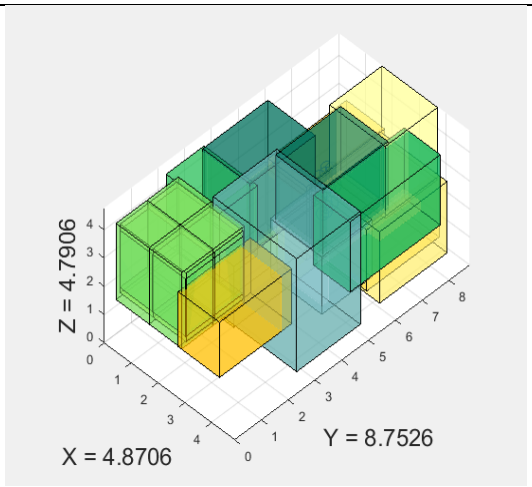
Container volume: 222.243



(b) overz_horizontal_4

(X, Y, Z): (3.666, 17.5517, 4)

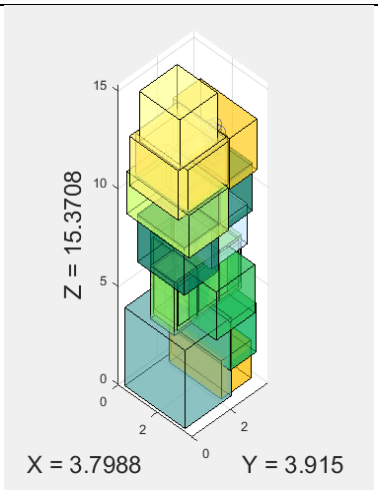
Container volume: 257.378



(c) overz_horizontal_5

(X, Y, Z): (4.8706, 8.7526, 4.7906)

Container volume: 204.225

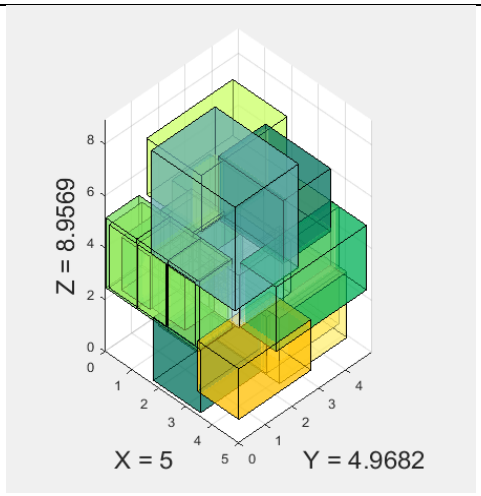


(d) overz_vertical_4

(X, Y, Z): (3.7988, 3.915, 15.3708)

Container volume: 228.599

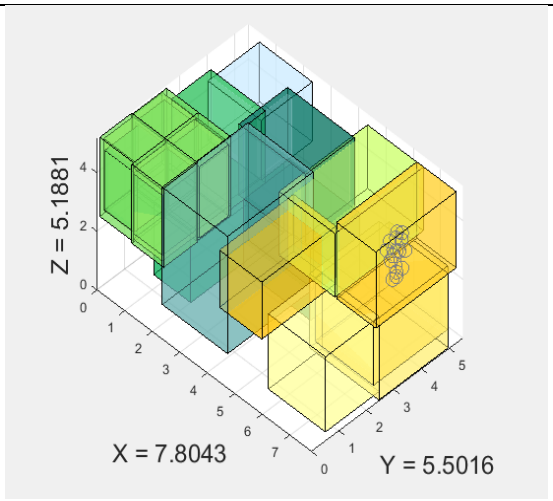
Figure 6: 10 layout designs for 14 task volumes



(e) overz_vertical_5

(X, Y, Z): (5, 4.9682, 8.9569)

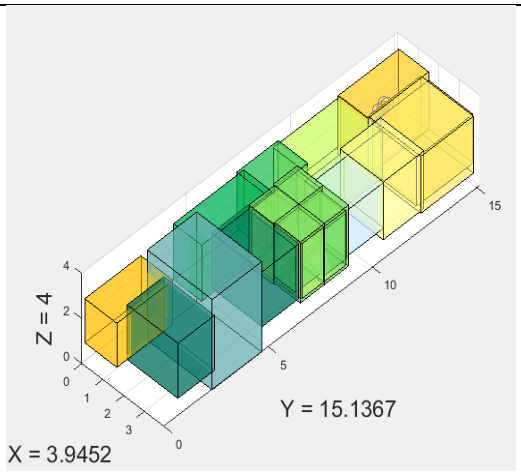
Container volume: 222.498



(f) solidz_cuboid

(X, Y, Z): (7.8043, 5.5016, 5.1881)

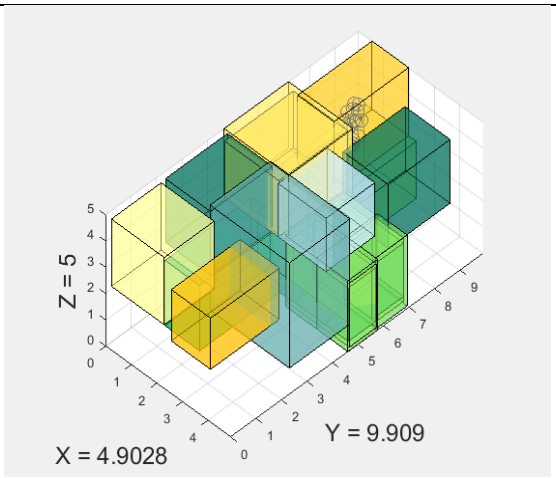
Container volume: 222.757



(g) solidz_horizontal_4

(X, Y, Z): (3.9452, 15.1367, 4)

Container volume: 238.870

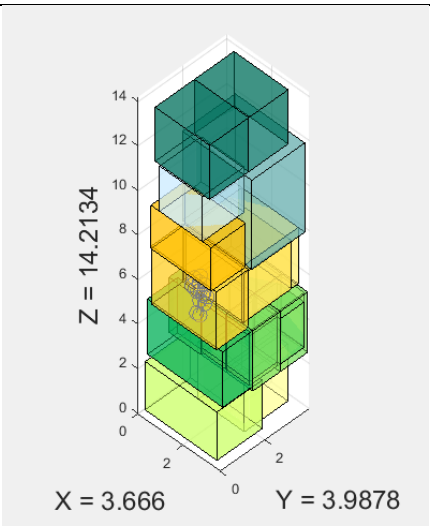


(h) solidz_horizontal_5

(X, Y, Z): (4.9028, 9.909, 5)

Container volume: 242.909

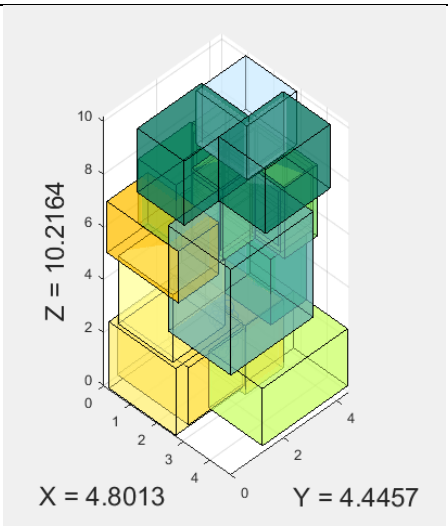
Figure 6, continued



(i) solidz_vertical_4

(X, Y, Z): (3.666, 3.9878, 14.2134)

Container volume: 207.790



(j) solidz_vertical_5

(X, Y, Z): (4.8013, 4.4457, 10.2164)

Container volume: 218.070

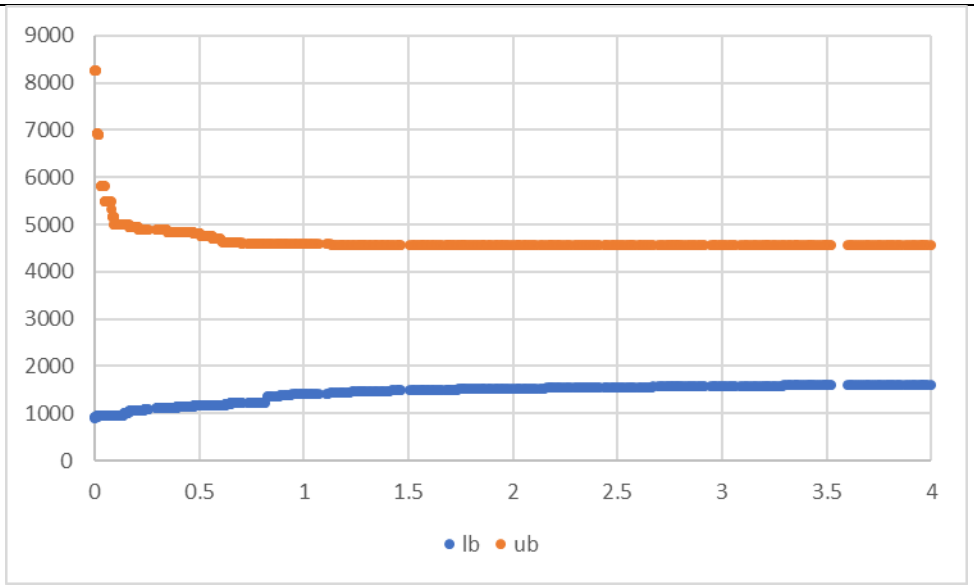
Figure 6, continued

Table 8: Optimality gap for large-scale problems

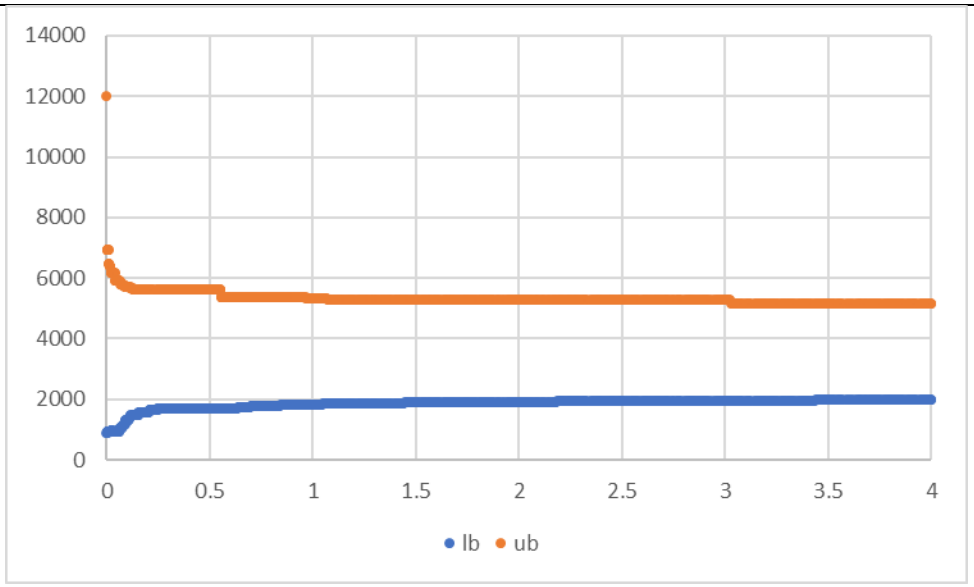
Layout Design	Optimality Gap
overz_cuboid	65.0%
overz_horizontal_4	61.5%
overz_horizontal_5	67.0%
overz_vertical_4	61.2%
overz_vertical_5	61.2%
solidz_cuboid	62.6%
solidz_horizontal_4	60.5%
solidz_horizontal_5	66.0%
solidz_vertical_4	59.4%
solidz_vertical_5	63.4%

Looking at Table 8, it can be noticed that the optimality gaps are rather high after 4 hours for all layout designs. To observe the rate at which the optimality gap decreased

throughout the 4-hour time period for each layout design, lower and upper bounds on the optimal objective values are plotted in Figure 7.

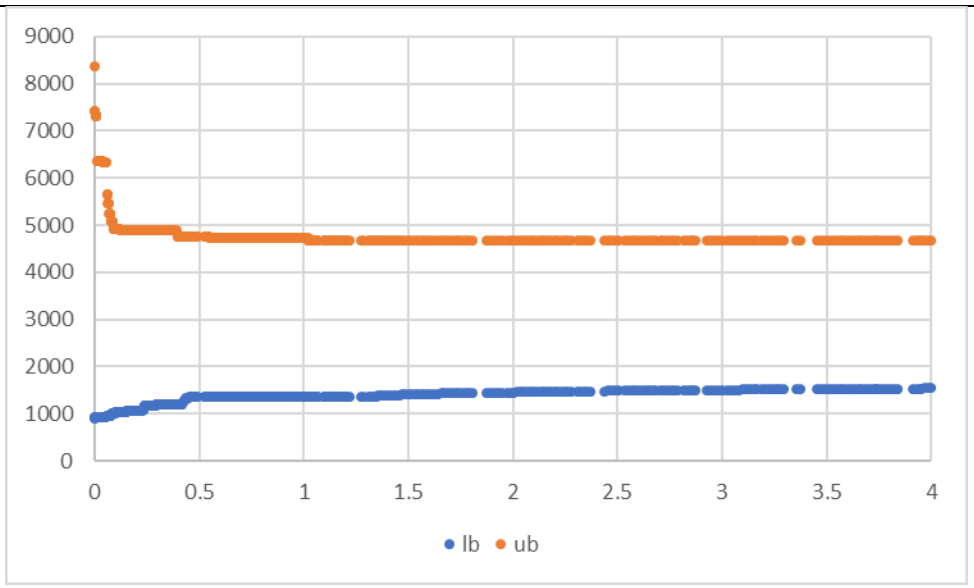


(a) overz_cuboid

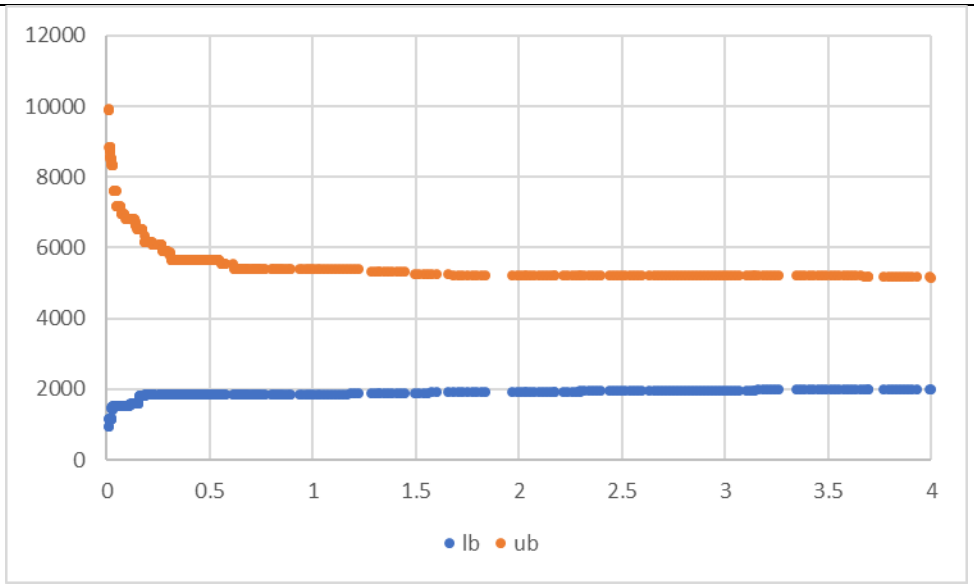


(b) overz_horizontal_4

Figure 7: Lower and upper bounds on the optimal objective values for 10 layout designs

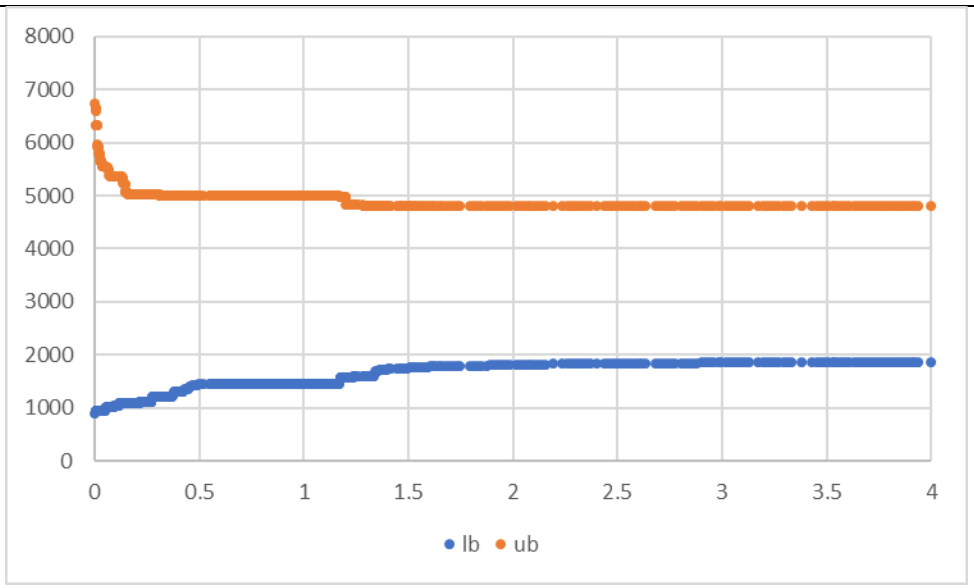


(c) overz_horizontal_5

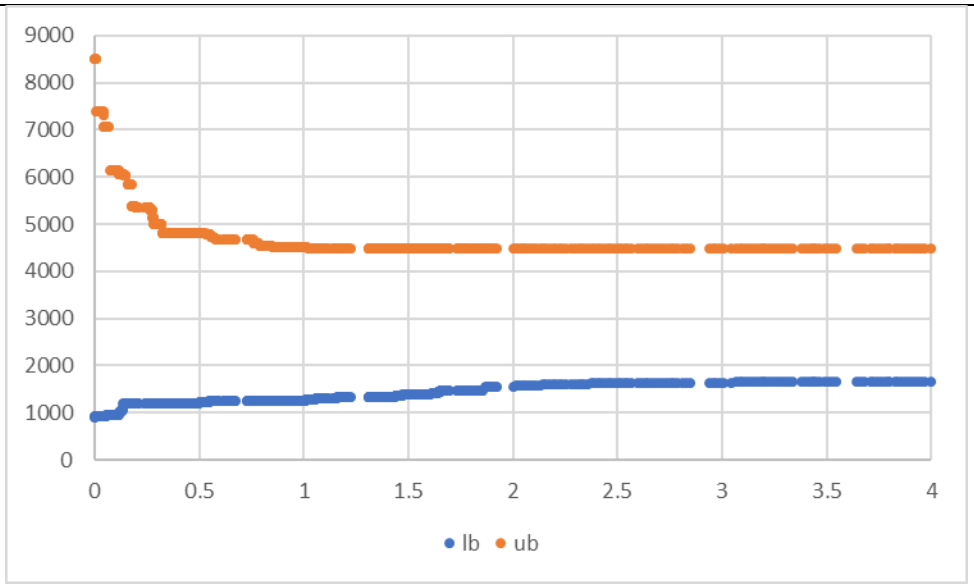


(d) overz_vertical_4

Figure 7, continued

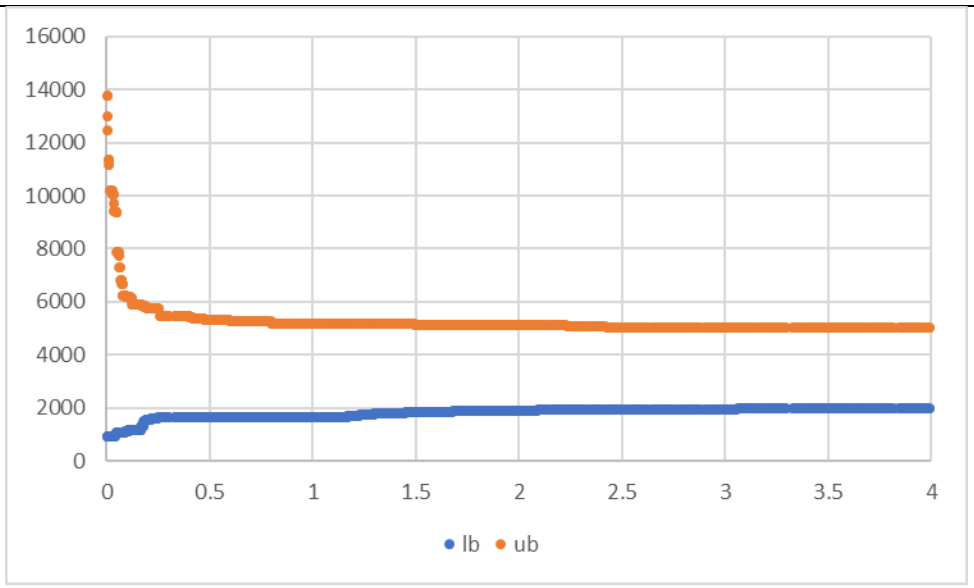


(e) overz_vertical_5

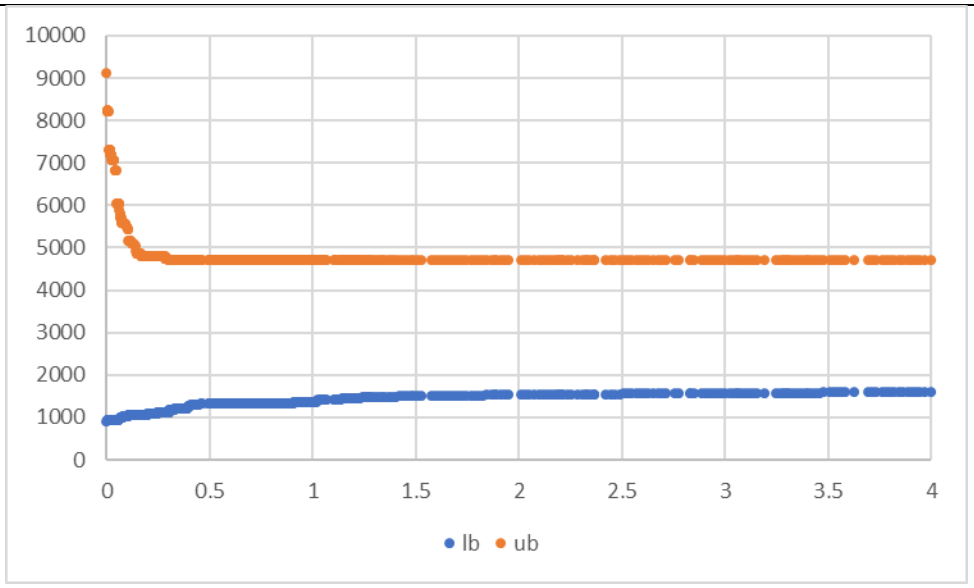


(f) solidz_cuboid

Figure 7, continued

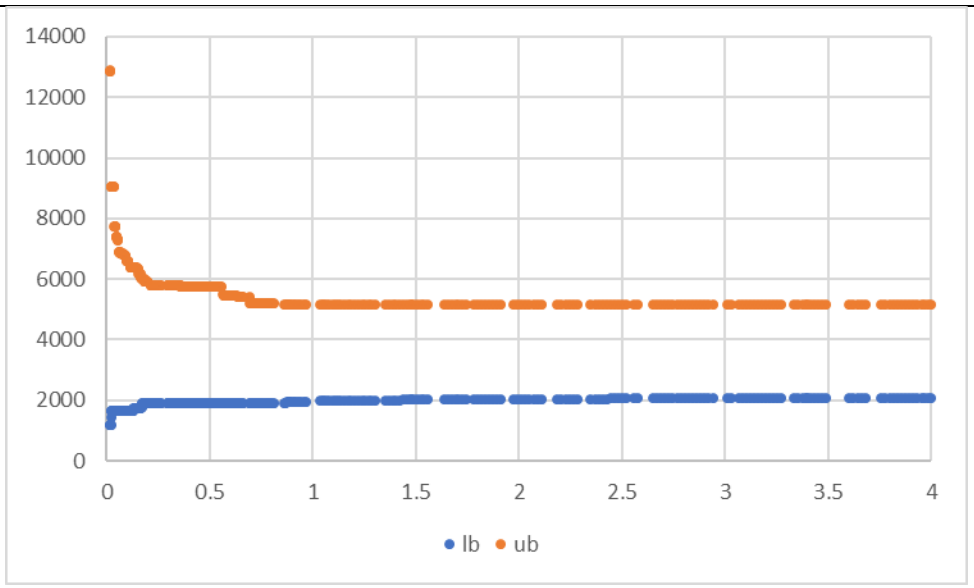


(g) solidz_horizontal_4

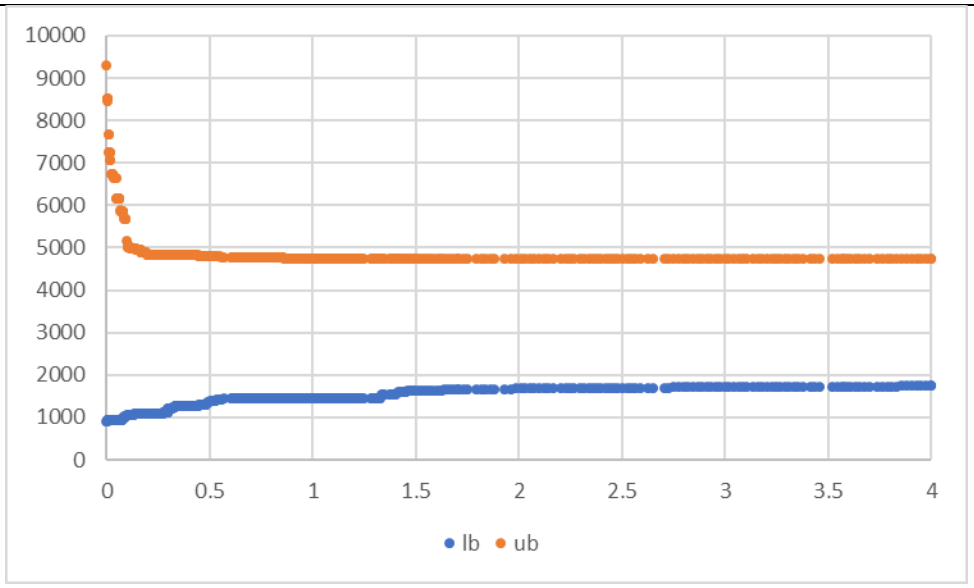


(h) solidz_horizontal_5

Figure 7, continued



(i) solidz_vertical_4

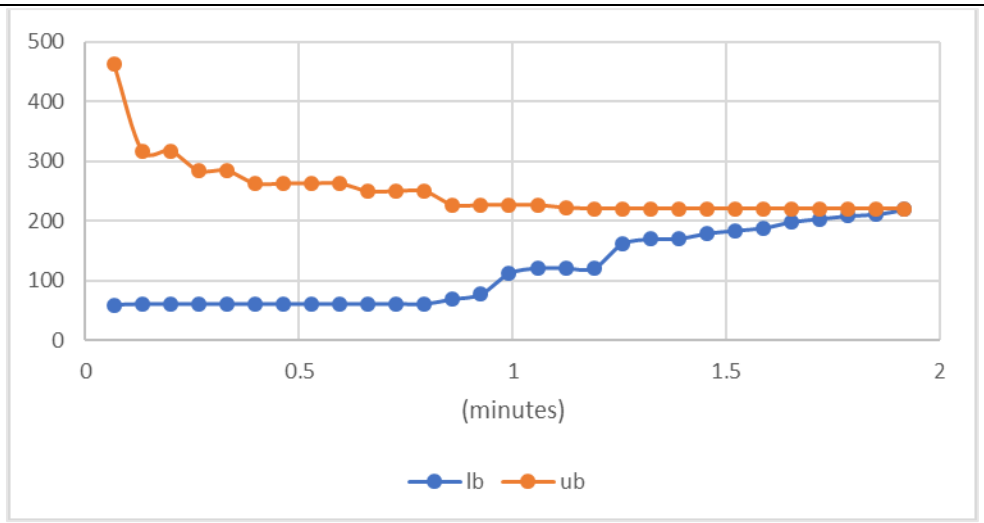


(j) solidz_vertical_5

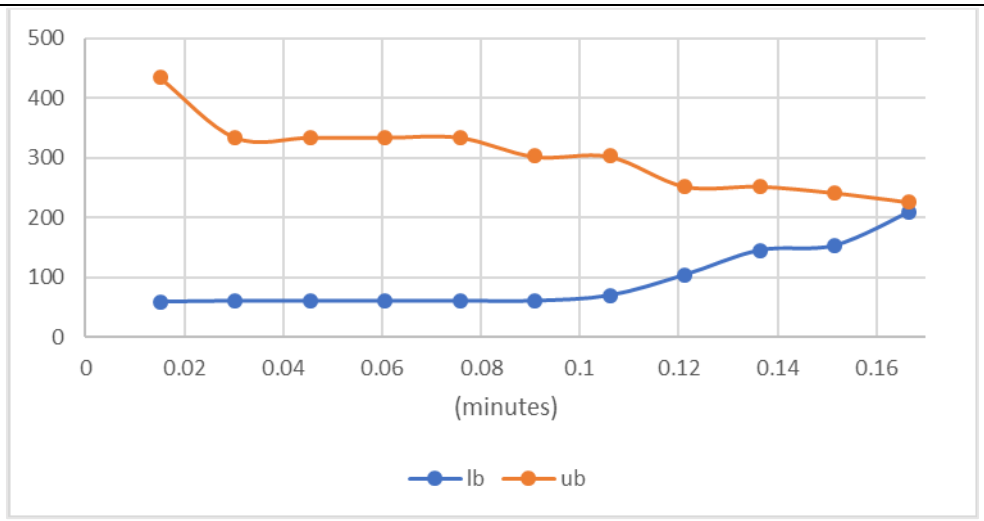
Figure 7, continued

Observing that optimality gaps for large-scale problems remain sizeable, we also plotted the optimality gaps for small-scale problems to see the pattern of lower and upper

bounds on the optimal objective values. Although there is no theoretical basis, the patterns of optimality gaps for small-scale test problems may help us conjecture how the optimality gaps of large-scale problems will behave in sustained runs.

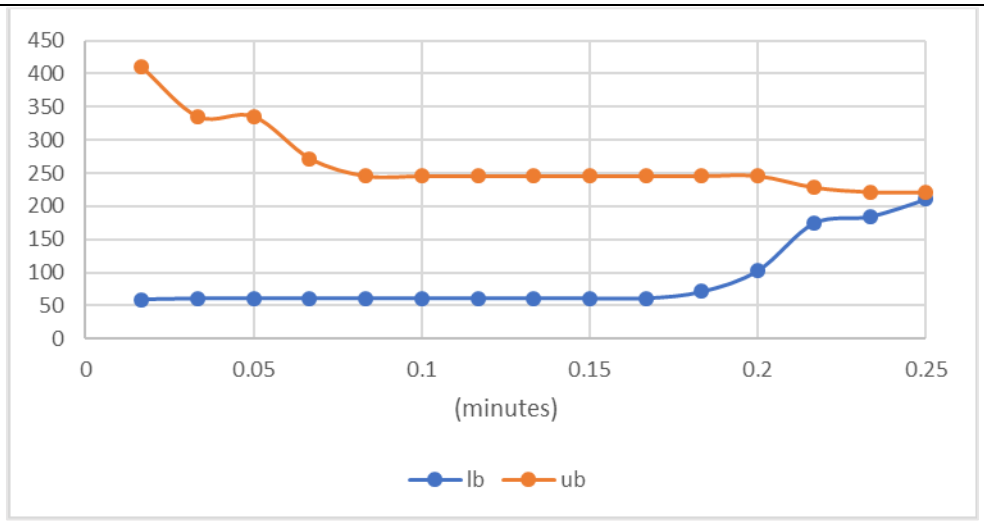


(a) overz_cuboid: small-scale

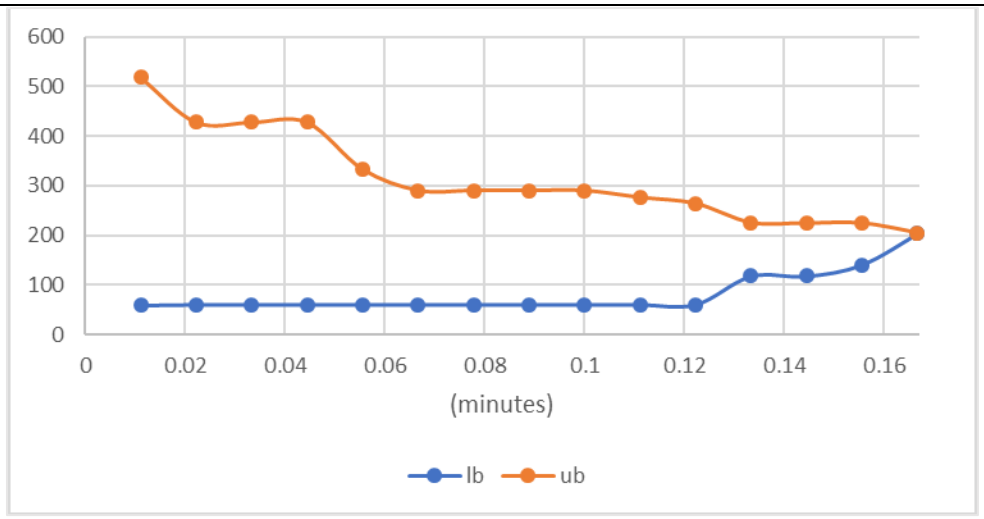


(b) overz_horizontal_4: small-scale

Figure 8: Lower and upper bounds on the optimal objective values for small-scale problems

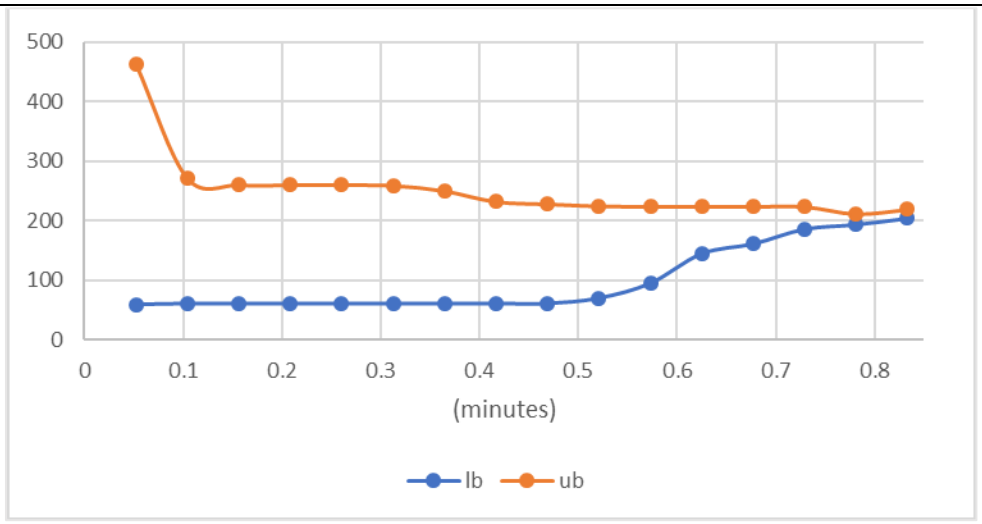


(c) overz_horizontal_5: small-scale

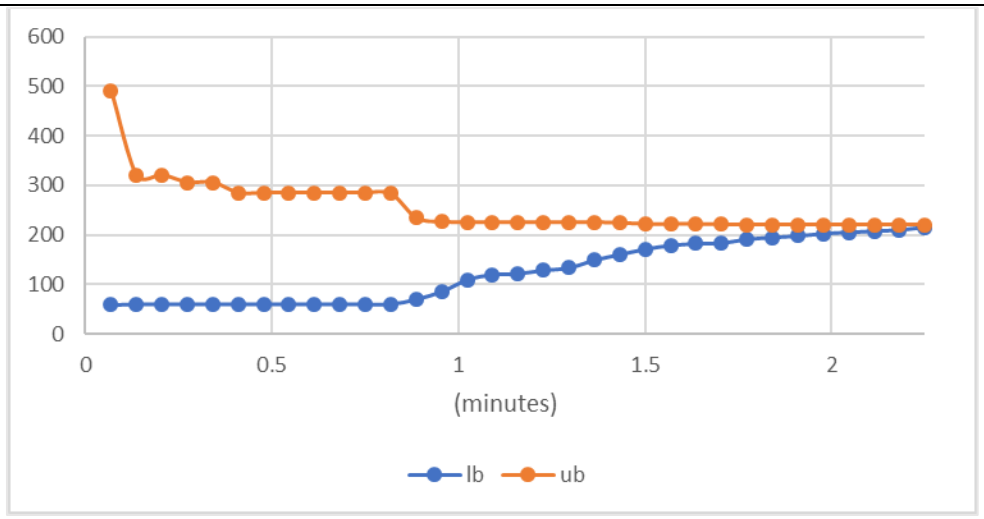


(d) overz_vertical_4: small-scale

Figure 8, continued

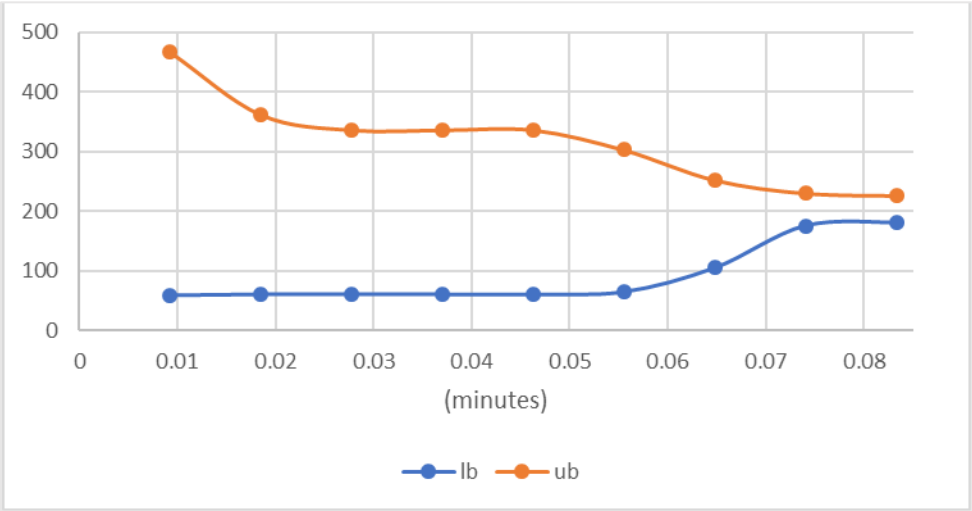


(e) overz_vertical_5: small-scale

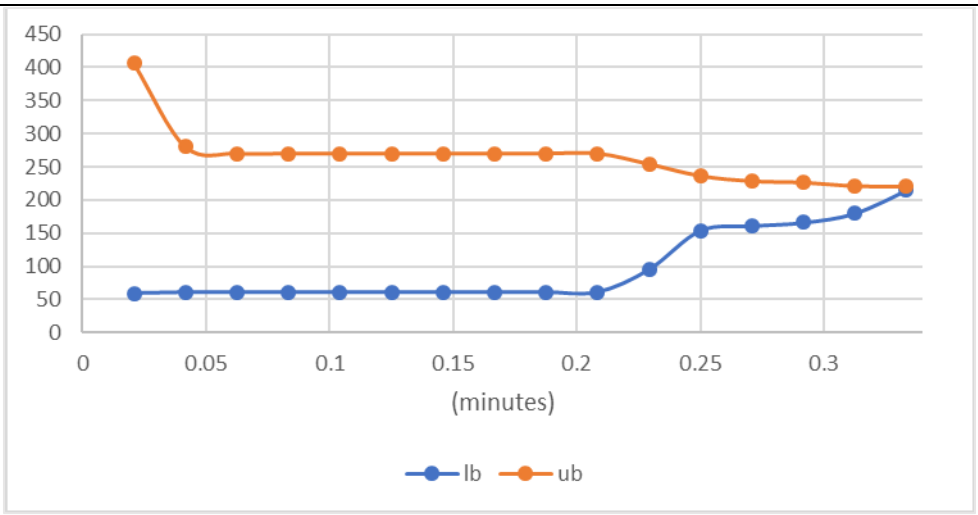


(f) solidz_cuboid: small-scale

Figure 8, continued

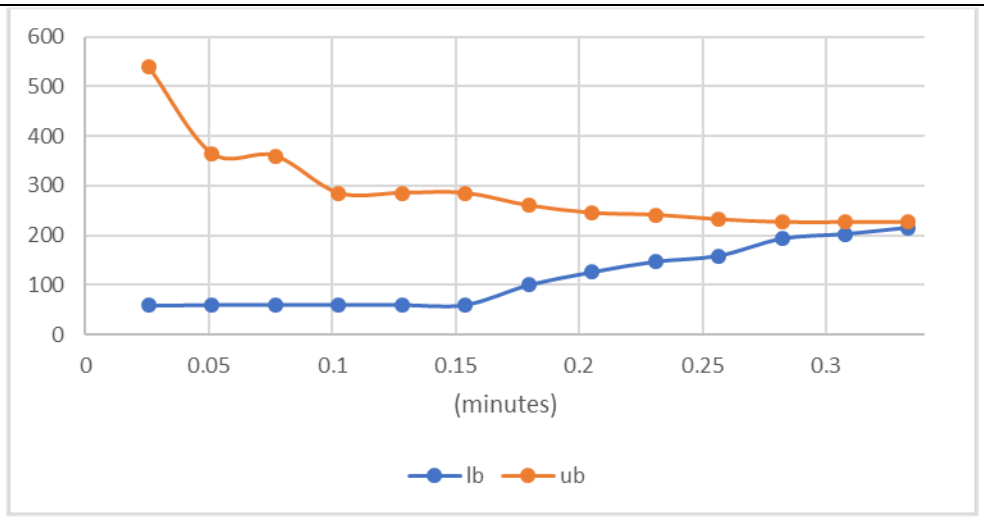


(g) solidz_horizontal_4: small-scale

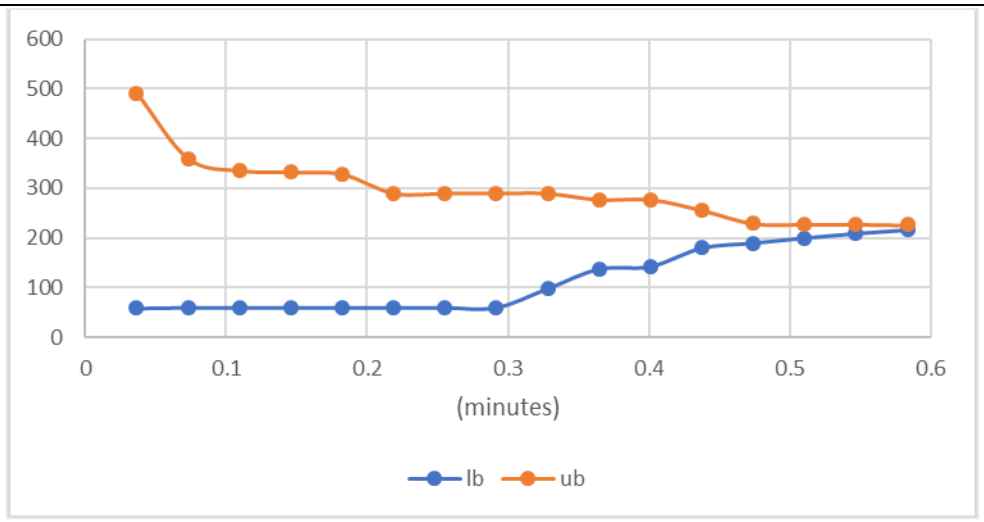


(h) solidz_horizontal_5: small-scale

Figure 8, continued



(h) solidz_vertical_4: small-scale



(i) solidz_vertical_5: small-scale

Figure 8, continued

From the plots, we can observe that the upper bound would remain relatively the same throughout the optimization once it is stabilized while the lower bound gets closer to the upper bound. Before closing the chapter, problem sizes between small- and large-scale problems for the base model are compared in Table 9.

Table 9: Comparison of problem sizes of small- and large-scale problems

	Small-Scale	Large-Scale
# of cuboids	7	24
# of inequality constraints	867	10914
# of equality constraints	22	73
# of constraints	889	10987
# of binary variables	304	3891
# of continuous variables	66	423
# of variables	370	4314

Resulting long computation times for solving OCPP motivates us to further investigate potential model enhancements and reformulations to the MILP in order to reduce the required computational effort. In the next chapter, additional constraints to the base model will be introduced in an effort to improve model efficiency. Accordingly, in Chapter 6, an additional comparative computational study will be presented for the same small- and large-scale problems defined in this chapter in order to demonstrate the effectiveness of the enhancement effort.

CHAPTER 5: MODEL ENHANCEMENTS

As mentioned earlier, the two-dimensional packing problem is NP-complete. Noting that OCPP is a generalization of the three-dimensional packing problem, it is unlikely to find an optimal solution even for a moderately-sized problem. In this chapter, it is intended to improve the computational effort of the original model by introducing additional constraints, including equality and valid inequality constraints, as well as by reformulating certain constraints in the original model. Once these constraints are formally stated in this chapter, their computational performance will be presented in the next chapter. In specific, seven types of additional constraints are introduced. Six of these constraints can be applied to all layout designs, while the remaining constraint is only applicable for unbounded and vertical layout designs. A discussion as to why this is the case will be presented once this constraint is introduced.

The first constraint that will be considered is associated with the relative orientation between a pair of cuboids. As mentioned before, a cuboid can either have its width face or length face parallel to the x-axis. Recall that binary variables I_{ij}^{fg} for $f, g \in \{w, l\}$ represent the relative orientation between cuboids *i* and *j*. Since only one of four possible relative orientations is to choose, the following constraint can be added for each pair of *i* and *j*:

$$I_{ij}^{ww} + I_{ij}^{wl} + I_{ij}^{lw} + I_{ij}^{ll} = 1 \qquad \forall\, i, j \qquad (41)$$

This constraint is intended to restrict the solution space, hoping that it reduces the computational time by tightening the LP relaxation of the problem. This constraint is implied to hold true in the original model due to (6) – (17), but is not explicitly stated. Without this explicit constraint enforced, it is possible that unnecessary calculations can

be performed to implicitly enforce one of these orientation variables to be equal to 1. Once the binary orientation variable to be equal to 1 is determined in a branch of the solution process, it is no longer necessary for any other orientation variables to be considered in further calculations. A similar case exists for linearization constraints.

The second type of constraints that are introduced are associated with the overlay between a pair of cuboids. This set of valid inequality constraints are shown below:

$$p_{ij} + p_{ji} \leq 1 \qquad \forall\, i,j \qquad i < j \tag{42}$$

$$q_{ij} + q_{ji} \leq 1 \qquad \forall\, i,j \qquad i < j \tag{43}$$

$$r_{ij} + r_{ji} \leq 1 \qquad \forall\, i,j \qquad i < j \tag{44}$$

Enforcing the first constraint in this set limits one cuboid to be to the left of another cuboid for any given pair. Recall that p_{ij} is equal to 1 if cuboid *i* is to the left of cuboid *j*. If p_{ij} is equal to 1, then this implies that cuboid *j* cannot be to the left of cuboid *i*, forcing p_{ji} to equal 0. Enforcing the second constraint limits one cuboid to be in front of another cuboid for any given pair. Recall that q_{ij} is equal to 1 if cuboid *i* is in front of cuboid *j*. If q_{ij} is equal to 1, then this implies that cuboid *j* cannot be in front of cuboid *i*, forcing q_{ji} to equal 0. Enforcing the third constraint limits one cuboid to be above another cuboid for any given pair. Recall that r_{ij} is equal to 1 if cuboid *i* is above cuboid *j*. If r_{ij} is equal to 1, then this implies that cuboid *j* cannot be above cuboid *i*, forcing r_{ji} to equal 0. Intuitively, it makes sense to enforce these constraints since it is not possible for both indicator variables included in each constraint to be equal to 1. Similar to the previous constraint type, this set of constraints are implied to hold true in the original model since these constraints influence the calculation of reference points but were not formally stated.

The third constraint type is associated with orientation variables for a single cuboid. These constraints will be enforced to influence the orientation of cuboids that are identical to each other. The third type of constraint is shown below:

$$I_i^w \leq I_{i+1}^w \leq \cdots \leq I_{i+k-1}^w, \tag{45}$$

where *k* is the number of cuboids for a task with multiple cuboids. This constraint is only applicable for the case where a task volume has multiple cuboids. If the indicator variable for cuboid *i* is equal to 1 and it belongs to a task volume where there are multiple cuboids, then this constraint will force the proceeding cuboids after index *i* included in this set of cuboids to equal 1 as well. For example, suppose there is a task volume with 3 cuboids ($i \in \{1,2,3\}$). If I_1^w is equal to 1, then both I_2^w and I_3^w are equal to 1. Alternatively, if I_1^w is equal to 0, then both I_2^w and I_3^w don't necessarily have to equal 1. Without this constraint in place, it is possible for many combinations of orientation variables to be activated for a particular task volume. Enforcing this constraint helps reduce this number of combinations.

Alternative constraints to some of the original constraints will now be introduced. The first set of constraints are an alternative formulation in representing the distance between a pair of cuboids and are shown below:

$$c_i^x - c_j^x = d_{ij}^{x^+} - d_{ij}^{x^-} \qquad \forall\, i,j \quad i < j \tag{46}$$

$$c_i^y - c_j^y = d_{ij}^{y^+} - d_{ij}^{y^-} \qquad \forall\, i,j \quad i < j \tag{47}$$

$$c_i^z - c_j^z = d_{ij}^{z^+} - d_{ij}^{z^-} \qquad \forall\, i,j \quad i < j \tag{48}$$

$$d_{ij}^{x^+} \geq 0 \qquad \forall\, i,j \quad i < j \tag{49}$$

$$d_{ij}^{x^-} \geq 0 \qquad \forall\, i,j \quad i < j \tag{50}$$

$$d_{ij}^{y^+} \geq 0 \qquad \forall\, i,j \quad i < j \tag{51}$$

$$d_{ij}^{y^-} \geq 0 \qquad \forall\, i,j \quad i < j \tag{52}$$

$$d_{ij}^{z^+} \geq 0 \qquad \forall\, i,j \quad i < j \tag{53}$$

$$d_{ij}^{z^-} \geq 0 \qquad \forall\, i,j \quad i < j \tag{54}$$

Accordingly, d_{ij} in the objective function is replaced by $d_{ij}^{x^+} + d_{ij}^{x^-} + d_{ij}^{y^+} + d_{ij}^{y^-} + d_{ij}^{z^+} + d_{ij}^{z^-}$. Originally, pairwise distance measurements required $4N(N-1)$ constraints to be included into the original model. With this alternative approach, $3N(N-1)$ inequality and $3N(N-1)/2$ equality constraints are necessary. In the alternative formulation, it can be observed that additional pairwise distance decision variables are necessary for these constraints to be valid. In particular, $3N(N-1)$ decision variables are required in this alternative reformulation, while $N(N-1)/2$ decision variables are required in the original formulation. These constraints will calculate the l_1 norm distance between respective center points of cuboids *i* and *j*. The purpose of including positive and negative distance variables is to represent the magnitude of the difference of center points in each dimension. In conjunction with the revised objective function, one of these two variables will be forced to be zero. For example, suppose that $c_i^x = 1$ and $c_j^x = 5$. The difference between these points in the x-dimension is -4 = $(c_i^x - c_j^x)$. Since the nonnegative variables representing the magnitudes in the positive and negative sides, respectively, are included in the objective function, minimization will force one of them to be zero. As a result, $d_{ij}^{x^+} = 0$ and $d_{ij}^{x^-} = 4$.

The second set of constraints is another alternative formulation for measuring distances between a pair of cuboids and is presented below:

$$d_{ij}^{x} \geq c_i^{x} - c_j^{x} \qquad \forall\, i,j \quad i < j \tag{55}$$

$$d_{ij}^{x} \geq c_j^{x} - c_i^{x} \qquad \forall\, i,j \quad i < j \tag{56}$$

$$d_{ij}^{y} \geq c_i^{y} - c_j^{y} \qquad \forall\, i,j \quad i < j \tag{57}$$

$$d_{ij}^{y} \geq c_j^{y} - c_i^{y} \qquad \forall\, i,j \quad i < j \tag{58}$$

$$d_{ij}^{z} \geq c_i^{z} - c_j^{z} \qquad \forall\, i,j \quad i < j \tag{59}$$

$$d_{ij}^{z} \geq c_j^{z} - c_i^{z} \qquad \forall\, i,j \quad i < j \tag{60}$$

$$d_{ij}^{x} \geq 0 \qquad \forall\, i,j \quad i < j \tag{61}$$

$$d_{ij}^{y} \geq 0 \qquad \forall\, i,j \quad i < j \tag{62}$$

$$d_{ij}^{z} \geq 0 \qquad \forall\, i,j \quad i < j \tag{63}$$

Accordingly, d_{ij} in the objective function is replaced by $d_{ij}^{x} + d_{ij}^{y} + d_{ij}^{z}$. As mentioned before, the original model required $4N(N-1)$ inequality constraints and $N(N-1)/2$ decision variables for measuring pairwise distances between cuboids, while this alternative representation requires $3N(N-1)$ inequality constraints and $3N(N-1)/2$ decision variables. For all cuboid pairs, it is guaranteed that one of the differences between the center points for a pair of cuboids is non-negative. It is also possible for the difference between the same pair of cuboids to be negative. However, since non-negativity constraints are enforced for the distance decision variables, the positive difference will only be considered in any solution. Since the problem of interest is a minimization problem, the minimum distance that is calculated between a pair of cuboids for any dimension will become the value of the distance decision variable, respective to each dimension, while enforcing that this distance must be non-negative.

The third set of constraints are an alternative formulation to linearization constraints that are necessary for the non-linear relationship between orientation variables. In the original model, linearization constraints were included for all pairs of cuboids with no predicate, which results in redundant constraints to be expressed (i.e., any given pair of cuboids will be considered twice). These constraints allow for the linearization constraints to only be considered when index *i* is less than *j*, and then introducing additional equality constraints to ensure that all indicator variables are considered in the linearization technique. This set of constraints are shown below:

$$I_{ij}^{ww} \leq I_i^w \qquad \forall\, i,j \quad i<j \qquad (64)$$

$$I_{ij}^{ww} \leq I_j^w \qquad \forall\, i,j \quad i<j \qquad (65)$$

$$I_{ij}^{ww} \geq I_i^w + I_j^w - 1 \qquad \forall\, i,j \quad i<j \qquad (66)$$

$$I_{ij}^{wl} \leq I_i^w \qquad \forall\, i,j \quad i<j \qquad (67)$$

$$I_{ij}^{wl} \leq 1 - I_j^w \qquad \forall\, i,j \quad i<j \qquad (68)$$

$$I_{ij}^{wl} \geq I_i^w + \left(1 - I_j^w\right) - 1 \qquad \forall\, i,j \quad i<j \qquad (69)$$

$$I_{ij}^{lw} \leq 1 - I_i^w \qquad \forall\, i,j \quad i<j \qquad (70)$$

$$I_{ij}^{lw} \leq I_j^w \qquad \forall\, i,j \quad i<j \qquad (71)$$

$$I_{ij}^{lw} \geq (1 - I_i^w) + I_j^w - 1 \qquad \forall\, i,j \quad i<j \qquad (72)$$

$$I_{ij}^{ll} \leq 1 - I_i^w \qquad \forall\, i,j \quad i<j \qquad (73)$$

$$I_{ij}^{ll} \leq 1 - I_j^w \qquad \forall\, i,j \quad i<j \qquad (74)$$

$$I_{ij}^{ll} \geq (1 - I_i^w) + (1 - I_j^w) + 1 \qquad \forall\, i,j \quad i<j \qquad (75)$$

$$I_{ij}^{ww} - I_{ji}^{ww} = 0 \qquad \forall\, i,j \quad i<j \qquad (76)$$

$$I_{ij}^{wl} - I_{ji}^{lw} = 0 \qquad \forall\, i,j \quad i<j \qquad (77)$$

$$I_{ij}^{lw} - I_{ji}^{wl} = 0 \qquad \forall\, i,j \quad i < j \tag{78}$$

$$I_{ij}^{ll} - I_{ji}^{ll} = 0 \qquad \forall\, i,j \quad i < j \tag{79}$$

In the original model, $12N(N-1)$ inequality constraints are required for linearization of the orientation variables while this alternative formulation requires $6N(N-1)$ inequality and $2N(N-1)$ equality constraints.

As mentioned at the beginning of the chapter, we will introduce an additional type of constraints that are only applicable to unbounded and vertical layout designs. Within [14], it is possible to force certain cuboids to have a certain orientation before the problem is solved based on design requirements, and then one can attempt to find an optimal solution to the problem. For this problem, it is possible to force a certain cuboid to have its width face along one of the sides of the container, for example, the plane defined by $y = 0$. This set of equality constraints is shown below:

$$I_i^w = 1 \tag{80}$$

$$I_{ij}^{lw} = 0 \qquad \forall\, j \quad i \neq j \tag{81}$$

$$I_{ij}^{ll} = 0 \qquad \forall\, j \quad i \neq j \tag{82}$$

$$I_{ij}^{ww} + I_{ij}^{wl} = 1 \qquad \forall\, j \quad i \neq j \tag{83}$$

Providing information about the orientation of a certain cuboid before the problem is solved reduces the solution space. Enforcing the first constraint in this set will restrict cuboid *i* to have its width face parallel to the x-axis. As a result of the first constraint being enforced, it is impossible for any relative orientation variable associated with cuboid *i's* length being parallel to the x-axis to be activated for any pair of cuboids and is enforced in the second and third constraints. Since only information about cuboid *i* is provided before the problem is solved, cuboid *j* is still permitted to have its width or

length parallel to the x-axis, However, only one of these instances are possible for cuboid j and is enforced in the fourth constraint. Forcing this restriction on orientation variables is intended to reduce computational time, however it is stated in [14] that it is not guaranteed that computational time will be reduced by enforcing certain orientations to exist.

This constraint is not applicable for horizontal layout designs because fixing the orientation of a cuboid can result in a non-optimal solution to the original problem. For example, suppose cuboid i's width is longer than its length. If it is desired to place cuboid i's width parallel to the x-axis, then this could force the y-axis of the module to be extended in order to pack cuboid i. On the other hand, if cuboid i's length were to be placed parallel to the x-axis, then it is likely that the y-axis of the module wouldn't have to be extended to the same degree as the previous situation, which results in a different layout solution. This constraint doesn't have the same type of restriction for unbounded and vertical layout designs. For the unbounded layout design, all dimensions of the module are variable and for the vertical layout design, the x- and y-axis of the module are fixed. Suppose that an optimal solution to the original problem has cuboid i's width parallel to the y-axis. Then, one can simply have the same solution by rotating the entire layout by 90 degrees, which results in cuboid i's width parallel to the x-axis while the rest of the cuboids are kept in the same relative locations.

For the sake of simplicity, (41) will be referred to as enhancement constraint (EC) 1, (42) – (44) will be considered as EC2, (45) will be considered as EC3, (46) – (54) will be considered as EC4, (55) – (63) will be considered as EC5, (64) – (79) will be considered as EC6, and (80) – (83) will be considered as EC7. In the next chapter, these

constraints will be tested to determine if they reduce the computational time to solve the MILP in reference to the computational study that was performed in Chapter 4.

CHAPTER 6: ENHANCED MODEL COMPUTATIONAL STUDY

In this chapter, a computational study is presented for the additional constraints that are proposed in Chapter 5 to investigate the effectiveness of those constraints in reducing computational efforts. Initially, in this computational study, each type of constraint is tested individually in the original model to determine the impact of the constraint on the overall computational experience using the same set of scenarios that were used for the computational study in Chapter 4. Based on the computational performance of the individual constraint type, we also investigate interactions between different types of those constraints. Accordingly, several combinations of different types are tested.

As in Chapter 4, 10 layout designs were considered in this comparative study. These layout designs consist of unbounded layout design, horizontal layout design with δ equal to 4 and 5, and vertical layout design with δ equal to 4 and 5, each of which either allows overlap along the z- axis or not.

6.1 Small-Scale Computational Study

In order to examine the efficacy of each additional constraint, EC1 – EC7 were individually tested for solving the 20 small-scale problems that were solved by the original formulation in Chapter 4. The MILP models were created using Matlab [32] and solved by calling a commercial MILP solver, Gurobi Optimizer 7.5 [33].

As described in Chapter 5, EC7 was tested only for unbounded and vertical layout designs while all other constraints were tested for each layout design. Since these constraints do no cut off the optimal solution, the optimal layouts remain the same as those produced by the original problem in Chapter 4. After finding optimal solutions to

all 20 scenarios for each layout design variant, time averages of 20 solution times are reported in Table 10.

Table 10: Average solution times for individual enhancements (sec)

Layout Design	EC						
	1	2	3	4	5	6	7
overz_cuboid	137.95	251.04	373.50	161.38	184.80	196.88	98.46
overz_horizontal_4	19.44	23.95	22.97	14.83	6.48	15.18	-
overz_horizontal_5	69.52	87.89	80.51	57.90	46.57	86.58	-
overz_vertical_4	34.65	41.47	35.65	29.75	23.80	37.81	21.97
overz_vertical_5	59.30	68.24	115.69	63.71	91.65	102.92	40.44
solidz_cuboid	127.69	270.93	304.86	148.77	167.98	112.13	91.60
solidz_horizontal_4	16.58	18.87	16.36	8.54	6.96	14.80	-
solidz_horizontal_5	46.34	68.83	66.40	45.34	36.93	76.79	-
solidz_vertical_4	34.81	34.25	35.58	29.65	25.45	36.84	17.98
solidz_vertical_5	64.28	91.41	85.99	64.86	61.96	81.40	36.25

To compare with solution times of the original formulation, percentage differences were computed and are displayed in Table 11. For EC7, the exercise gradient cuboid was selected to have its width parallel to the x-axis.

Table 11: Percentage difference for 10 layout designs and individual enhancement constraints

	EC						
	1	2	3	4	5	6	7
overz_cuboid	43.75	-2.36	-52.29	34.20	24.65	19.72	59.85
overz_horizontal_4	-19.51	-47.22	-41.17	8.84	60.15	6.67	-
overz_horizontal_5	29.49	10.86	18.34	41.27	52.77	12.18	-
overz_vertical_4	-3.00	-23.28	-5.97	11.56	29.24	-12.41	34.70
overz_vertical_5	19.97	7.91	-56.13	14.02	-23.69	-38.89	45.42
solidz_cuboid	58.10	11.09	-0.04	51.18	44.88	63.20	69.94
solidz_horizontal_4	-12.00	-27.48	-10.53	42.33	52.94	0.00	-
solidz_horizontal_5	39.31	9.86	13.03	40.62	51.62	-0.58	-
solidz_vertical_4	16.67	18.02	14.84	29.03	39.08	11.83	56.96
solidz_vertical_5	6.76	-32.60	-24.73	5.91	10.12	-18.08	47.42

In Table 11, green cells with positive values indicate better performances compared to that of the original formulation, while red cells with negative values represent worse performances. Observe that EC4 consistently outperformed the original formulation. Furthermore, EC7 uniformly displays significant improvement for applicable layout designs. EC5 also displays better performances than the original formulation except for vertical layout design with $\delta = 5$ when overlap along the z-axis is allowed. Observing that layout designs with and without overlap along the z-axis tend to have similar tendency in performance, results were aggregated with respect to overlap along the z-axis. Aggregated percentage improvements are displayed in Table 12.

Table 12*:* Percentage difference with model enhancements

Aggregated Layout Design	EC						
	1	2	3	4	5	6	7
Unbounded Layout Design	51.70	5.09	-23.34	43.61	35.86	43.81	65.44
Horizontal Layout Design ($\delta = 4$)	-15.93	-37.82	-26.57	24.79	56.72	3.49	-
Horizontal Layout Design ($\delta = 5$)	33.77	10.42	16.02	40.99	52.27	6.61	-
Vertical Layout Design ($\delta = 4$)	7.90	-0.40	5.56	21.23	34.69	1.02	47.03
Vertical Layout Design ($\delta = 5$)	13.60	-11.61	-40.99	10.11	-7.39	-28.86	46.38

For the unbounded layout design, which is an aggregation of overz_cuboid and solidz_cuboid, it was noticed that EC1, 4, 5 and 7 individually reduced the computational effort by at least 35.86%. Although EC2 did offer some improvement, it was not

convincing enough to pursue further testing. For this layout design category, the following combinations are considered for testing interactions:

- EC1, 4, 6, 7 (Combination 1)
- EC1, 5, 6, 7 (Combination 2)

Since EC4 and EC5 are alternative constraints to represent distance between cuboids, they were not considered together.

For the horizontal layout design, it was noticed that there were distinct differences in $\delta = 4$ and $\delta = 5$. EC1, 2, and 3 perform much better on average for the horizontal layout design with δ equal to 5. EC1 provided an average improvement of 8.92% for both horizontal layout cases, whereas EC2 and EC3 display worse performances. EC4 and EC5 individually reduced the computational effort by at least 24.79% in both cases. EC6 did not provide significant improvement for either case. For this layout design category, the following combinations are selected to be further investigated:

- EC1, 4 (Combination 3)
- EC1, 5 (Combination 4)
- EC1, 4, 6 (Combination 5)
- EC1, 5, 6 (Combination 6)

Even though EC6 did not provide an impressive improvement, it was included to check interaction effects.

For the vertical layout design, it was noticed that EC1 offered an improvement of at least 7.9% for both cases with $\delta = 4$ and 5. EC2 and EC3 seemed to have worsened the solution time compared to the original result for vertical layout designs. EC4 and EC5 both reduced the computational time by at least 13.65%. EC7 decreased the

computational time by at least 46.38%. For this layout design category, the following combinations of constraints were considered:

- EC1, 4, 7 (Combination 7)
- EC1, 5, 7 (Combination 8)

For each layout design category, a hypothesis test was performed for the two combinations that display the smallest average solution times. This was performed in order to determine if there is any statistically significant difference in solution times for a pair of combinations. The output of these tests may provide evidence as to which combination is better. Within this analysis, the average solution time and standard deviations for the model with and without model enhancements, as well as maximum and minimum solution times, will be presented. For each combination that includes EC7, the exercise gradient cuboid was selected to have its width parallel to the x-axis.

6.1.1 Unbounded Layout Design

For the unbounded layout design, combinations 1 and 2 were considered. These combinations were tested for both cases of the unbounded layout design, where overlap is allowed and overlap is not allowed along the z-axis. The average solution times were compared to the result that was generated in Chapter 4 to find how much of an improvement there is in computational effort. The output data for combination 1 and combination 2 is displayed below, as well as the output from the model without these additional constraints:

Table 13: Comparison between original model and enhanced model for unbounded layout design

	Average Solution Time	Standard Deviation	Max	Min
Original Model	275.00	254.12	1055.76	45.72
Combination 1	53.58	29.72	170.99	27.46
Combination 2	54.21	22.93	127.97	9.88

Combinations 1 and 2 provide a similar result such that their average solution times are almost the same and their standard deviations are slightly different. In order to determine if these two combinations provide the same result on average, a hypothesis test was performed. Let μ_1 denote the population mean of solution times for combination 1 and let μ_2 denote the population mean of solution times for combination 2. The hypothesis test that was performed is displayed below:

$$H_0: \mu_1 = \mu_2$$

$$H_a: \mu_1 \neq \mu_2$$

Before testing these hypotheses, an F-test was first performed in order to determine if the population standard deviations for both combinations can be assumed equal or not. Let σ_1 denote the population standard deviation of solution times for combination 1 and let σ_2 denote the population standard deviation of solution times for combination 2. The hypothesis test that was performed is displayed below:

$$H_0: \sigma_1 = \sigma_2$$

$$H_a: \sigma_1 \neq \sigma_2$$

With a p-value of 0.479 using Bonett's test and 0.551 using Levene's test for the F-test, it was assumed that the population standard deviations for both combinations are equal to one another. After performing the t-test using this assumption, a p-value of 0.916 was obtained which implies that there is no statistically significant difference between μ_1 and μ_2 and fail to reject the null hypothesis. Since there appears to be no difference statistically between combinations 1 and 2 solution times, combination 2 was selected to be tested for solving the large-scale problem in order to measure the change in optimality gap for the unbounded layout design since it had a lower sample standard deviation.

6.1.2 Horizontal Layout Design

For the horizontal layout design, combinations 3 - 6 were considered. These combinations were tested for both cases of the horizontal layout design where δ is equal to 4 and 5. The output data for combination 3, combination 4, combination 5, and combination 6 is displayed below, as well as the output from the model without these additional constraints where δ is equal to 4:

Table 14: Comparison between original model and enhanced model for horizontal layout design where δ is equal to 4

	Average Solution Time	Standard Deviation	Max	Min
Original Model	15.54	9.85	56.26	5.01
Combination 3	13.44	14.81	63.82	4.93
Combination 4	8.95	4.53	34.62	4.66
Combination 5	5.32	4.34	25.36	3.13
Combination 6	7.17	3.42	23.63	4.18

Combinations 5 and 6 have the least average solution time in this set of combinations. Let μ_5^4 denote the population mean of solution times for combination 5 and let μ_6^4 denote the population mean of solution times for combination 6 where δ is equal to 4. The hypothesis test that was performed is displayed below:

$$H_0: \mu_5^4 = \mu_6^4$$

$$H_a: \mu_5^4 \neq \mu_6^4$$

Let σ_5^4 denote the population standard deviation of solution times for combination 5 and let σ_6^4 denote the population standard deviation of solution times for combination 6 where δ is equal to 4. The hypothesis test that was performed is displayed below:

$$H_0: \sigma_5^4 = \sigma_6^4$$

$$H_a: \sigma_5^4 \neq \sigma_6^4$$

With a p-value of 0.640 using Bonett's test and 0.896 using Levene's test for the F-test, it was assumed that the population standard deviations for both combinations are equal to one another. After performing the t-test using this assumption, a p-value of 0.038 was obtained which implies that there is a statistically significant difference between σ_5^4 and σ_6^4, and the null hypothesis is not accepted if α = 0.05. Since there appears to be a statistical difference between combinations 5 and 6 where δ is equal to 4, combination 5 was selected to be tested for solving for the large-scale problem for the horizontal layout design where δ is equal to 4.

The output data for combination 3, combination 4, combination 5, and combination 6 is displayed below, as well as the output from the model without these additional constraints where δ is equal to 5:

Table 15: Comparison between original model and enhanced model for horizontal layout design where δ is equal to 5

	Average Solution Time	Standard Deviation	Max	Min
Original Model	87.47	100.58	431.28	9.24
Combination 3	52.85	40.01	184.94	7.29
Combination 4	42.44	31.45	118.53	9.61
Combination 5	44.91	32.68	127.95	3.85
Combination 6	46.87	52.08	238.85	5.54

Combinations 4 and 5 have the least average solution times in this set of combinations. Let μ_4^5 denote the population mean of solution times for combination 4 and let μ_5^5 denote the population mean of solution times for combination 5 where δ is equal to 5. The hypothesis test that was performed is displayed below:

$$H_0: \mu_4^5 = \mu_5^5$$

$$H_a: \mu_4^5 \neq \mu_5^5$$

Let σ_4^5 denote the population standard deviation of solution times for combination 4 and let σ_5^5 denote the population standard deviation of solution times for combination 5 where δ is equal to 5. The hypothesis test that was performed is displayed below:

$$H_0: \sigma_4^5 = \sigma_5^5$$

$$H_a: \sigma_4^5 \neq \sigma_5^5$$

With a p-value of 0.846 using Bonett's test and 0.768 using Levene's test for the F-test, it was assumed that the population standard deviations for both combinations are equal to each other. After performing the t-test using this assumption, a p-value of 0.732 was

obtained which implies that there is no significant difference between σ_4^5 and σ_5^5, and fail to reject the null hypothesis. Since there appears to be no difference statistically between combinations 4 and 5 where δ is equal to 5, combination 4 was selected to be tested for solving the large-scale problem for the horizontal layout design where δ is equal to 5.

6.1.3 Vertical Layout Design

For the vertical layout design, combinations 7 and 8 were considered. These combinations were tested for both cases with respective values of δ equal to 4 and 5. The output data for combination 7 where EC1, 4 and 7 are included and combination 8 where EC1, 5 and 7 are included is displayed below, as well as the output from the model without these additional constraints where δ is equal to 4:

Table 16: Comparison between original model and enhanced model for vertical layout design where δ is equal to 4

	Average Solution Time	Standard Deviation	Max	Min
Original Model	37.71	55.77	365.81	11.53
Combination 7	13.93	9.95	50.03	3.84
Combination 8	10.45	4.61	33.97	5.47

Let μ_7^4 denote the population mean of solution times for combination 7 and let μ_8^4 denote the population mean of solution times for combination 8 where δ is equal to 4. The hypothesis test that was performed is displayed below:

$$H_0: \mu_7^4 = \mu_8^4$$

$$H_a: \mu_7^4 \neq \mu_8^4$$

Let σ_7^4 denote the population standard deviation of solution times for combination 7 and let σ_8^4 denote the population standard deviation of solution times for combination 8 where δ is equal to 4. The hypothesis test that was performed is displayed below:

$$H_0: \sigma_7^4 = \sigma_8^4$$

$$H_a: \sigma_7^4 \neq \sigma_8^4$$

Observing a small p-value of 0.026 using Bonett's test and 0.000 using Levene's test for the F-test, it was assumed that the population standard deviations for both combinations are not equal to one another. After performing the t-test using this assumption, a p-value of 0.048 was obtained which implies that there is a statistically significant difference between σ_7^4 and σ_8^4 where α = 0.05. As a result, combination 8 was selected to be tested for solving the large-scale problem for the vertical layout design where δ is equal to 4.

The output data for combination 7 and combination 8 is displayed below, as well as the output from the model without these additional constraints δ is equal to 5:

Table 17: Comparison between original model and enhanced model for vertical layout design where δ is equal to 5

	Average Solution Time	Standard Deviation	Max	Min
Original Model	71.52	71.28	315.99	23.11
Combination 7	31.64	17.50	92.86	5.36
Combination 8	29.95	23.54	99.37	7.34

Let μ_7^5 denote the population mean of solution times for combination 7 and let μ_8^5 denote the population mean of solution times for combination 8 where δ is equal to 5. The hypothesis test that was performed is displayed below:

$$H_0: \mu_7^5 = \mu_8^5$$

$$H_a: \mu_7^5 \neq \mu_8^5$$

Let σ_7^5 denote the population standard deviation of solution times for combination 7 and let σ_8^5 denote the population standard deviation of solution times for combination 8 where δ is equal to 5. The hypothesis test that was performed is displayed below:

$$H_0: \sigma_7^5 = \sigma_8^5$$

$$H_a: \sigma_7^5 \neq \sigma_8^5$$

With a p-value of 0.259 using Bonett's test and 0.018 using Levene's test for the F-test, it was difficult to determine which assumption to follow. Instead, two t-tests were performed where both assumptions were considered to see if different conclusions were drawn. After performing the t-test where equal population standard deviations are assumed, a p-value of 0.717 was obtained which implies that there is no statistically significant difference between σ_7^5 and σ_8^5 and fail to reject the null hypothesis. After performing the t-test where equal population standard deviations are not assumed, a p-value of 0.717 was obtained which implies that there is no statistically significant difference between σ_7^5 and σ_8^5 and fail to reject the null hypothesis. Comparing the outcomes from both of these tests, it was concluded that there was no statistically significant difference between μ_7^5 and μ_8^5. As a result, combination 7 was selected to be tested for solving the large-scale problem for the vertical layout design where δ is equal to 5 since it had a lower sample standard deviation.

In Table 18, the solution times of selected combinations are compared with those of models without incorporating the above enhancements.

Table 18: Percentage difference without and with enhancements for small-scale problem

Aggregated Layout Design	Average Solution Time without Enhancements	Average Solution Time with Enhancements	Percentage Difference
Unbounded Layout Design	275.00	54.21	80.29%
Horizontal Layout Design (δ = 4)	15.54	5.32	65.77%
Horizontal Layout Design (δ = 5)	87.47	42.44	51.48%
Vertical Layout Design (δ = 4)	37.71	10.45	72.29%
Vertical Layout Design (δ = 5)	71.52	31.64	55.76%
Average	97.45	28.81	65.12%

On average, solution time is improved by 65.12% for all combinations that are included in each aggregated layout design.

Although combinations of constraints can be created to reduce computational effort, it is possible that a single EC could outperform the selected combination of ECs. In order to verify whether this is the case or not, the EC that provided the maximum improvement is compared to the combination that was selected for an aggregated layout design. The hypotheses tested are displayed below:

$$H_0: \mu_{combo} = \mu_{EC}$$

$$H_a: \mu_{combo} < \mu_{EC}$$

The table below displays the result of the hypothesis test for each aggregated layout design.

Table 19: Comparison between individual EC and combination of ECs

	Unbounded Layout Design	Horizontal Layout Design ($\delta = 4$)	Horizontal Layout Design ($\delta = 5$)	Vertical Layout Design ($\delta = 4$)	Vertical Layout Design ($\delta = 5$)
EC	7	5	5	7	7
Combo	2	5	4	8	7
Test Result	Reject ($p = 0.000$)	Reject ($p = 0.032$)	Fail to reject ($p = 0.468$)	Reject ($p = 0.000$)	Reject ($p = 0.034$)

Looking at the result in Table 19, it can be concluded that EC5 (standard deviation = 42.97) provides the same computational benefit, with regards to solution time, as combination 4 (standard deviation = 31.45) for the aggregated horizontal layout design with $\delta = 5$, on average. Other layout design cases display that the combination outperformed the implementation of a single EC. Although the horizonal layout design with $\delta = 5$ is inconclusive, we still recommend applying combination 4 for the sake of consistency.

6.2 Large-Scale Computational Study

As a result of comparing various combinations of constraints in the small-scale computational study, it was then possible to use a selection of these combinations in order to measure by how much the optimality gap is reduced for a larger sized problem. The large-scale problem consists of twenty-four gradients cuboids, which is associated with fourteen task volumes. For the single-dimensional packing optimization, the value of δ was selected as 4 and 5, resulting in two layouts for both layout designs. Weighting factor α in the objective function was set to 0.5 to obtain solutions from the models to be evaluated. In total, 10 layout designs were considered, as displayed in Table 5. The MILP models were created using Matlab [32] and solved by Gurobi Optimizer 7.5 [33]. For

testing purposes, the solver time was set to 4 hours. For each combination that includes EC7, the exercise gradient cuboid was selected to have its width face parallel to the x-axis. The following combinations were tested for all layout design categories as a result of the small-scale study:

- Unbounded Layout Designs: Combination 1
- Horizontal Layout Designs (δ equal to 4): Combination 5
- Horizontal Layout Designs (δ equal to 5): Combination 4
- Vertical Layout Designs (δ equal to 4): Combination 8
- Vertical Layout Designs (δ equal to 5): Combination 7

Table 20: Optimality gap without and with enhancements for large-scale problem

Layout Design	Optimality Gap without Enhancements	Optimality Gap with Enhancements	Difference
overz_cuboid	65.0%	52.1%	12.9%
overz_horizontal_4	61.5%	53.5%	8.0%
overz_horizontal_5	67.0%	59.8%	7.2%
overz_vertical_4	61.2%	51.0%	10.2%
overz_vertical_5	61.2%	51.0%	10.2%
solidz_cuboid	62.6%	51.9%	10.7%
solidz_horizontal_4	60.5%	55.4%	5.1%
solidz_horizontal_5	66.0%	62.7%	3.3%
solidz_vertical_4	59.4%	49.3%	10.1%
solidz_vertical_5	63.4%	47.7%	15.7%

As displayed in Table 20, combinations uniformly improved the optimality gaps across 10-large scale problems. On average, the optimality gap after a 4-hour run was improved by 14.92%, where maximum improvement is observed for solidz_vertical_5 by 15.7% and minimum improvement is observed for solidz_horizontal_5 by 3.3%.

CHAPTER 7: CONCLUSION

In this thesis research, the overlap cuboid packing problem (OCPP) was considered. This variant of the packing problem is unique in the sense that items are allowed to overlap to a certain extent. This provides the advantage to further minimize the volume of the container since items are allowed to share space. For this research, the OCPP was formulated using a mixed-integer linear program (MILP) to find an arrangement of gradient cuboids that represent volumes of tasks that are performed by astronauts inside of a spacecraft module. A bi-objective function was proposed to accommodate for two performance criteria, which include adjacency requirements between gradient cuboids and the size of the module.

After the MILP was formulated, a numerical study was executed to demonstrate the model's ability to produce different layout results corresponding to certain design requirements and priorities on randomly generated test problems as well as a full-scale problem. Noting that OCPP is NP-hard, the computational requirements in solving the problem can be formidable as evidenced by the numerical results for the full-scale problem. To alleviate such computational burden, it was of interest to further investigate model enhancements and reformulations in order to reduce the amount of computational effort. This endeavor resulted in seven sets of constraints.

An additional numerical study was performed where the additional constraints were initially tested individually using the small-scale problem. This was performed in order to determine which constraints, when included individually, are most beneficial to the solution of the MILP. Following this, various combinations of promising constraints were constructed for unbounded, horizontal, and vertical layout designs and tested on the

randomly generated problems to determine if further improvement could be achieved. As a result, it was observed that different combinations customized for each layout design provide improvement in computational efficiency. These combinations were applied to the full-scale problem, where it was observed that the MILP with those combinations produced smaller optimality gaps after a 4-hour run than optimality gaps produced without them for all layout designs

OCPP is applicable to a wide range of applications, especially in habitat-related architectural design such as interior of a home with a small volume, jail cells, and quarters of a submarine. There are some avenues where future studies can be conducted. The model can accommodate additional constraints such as structural integrity (e.g., impact), weight balancing, and utility layout (i.e., water supply, food storage, waste, etc.). Besides these problem-specific constraints, additional valid inequalities can be incorporated in an effort to further improve the computational efficiency. Also, future research includes creating a "feedback loop" between the physical and psycho-physical properties of the OCPP for astronauts. Creating a link between these two properties helps accommodate the physical needs of astronauts, as well as their mental needs. Incorporating this into the optimization allows for changes that are made to either property to be accounted for in the other where these changes are driven by the astronaut's preference.

REFERENCES

[1] S. S. Thaxton, M. Chen, S. Hsiang, C. Lim, J. Meyers, and S. Wald, “Spacecraft optimization layout and volume (SOLV): Development of a model to assess habitable volume,” *Proceedings of IEEE Aerospace Conference,* Big Sky, MT, Mar. 2017.

[2] N. Z. Hu, H. L. Li, and J. F. Tsai, "Solving packing problems by a distributed global optimization algorithm," *Mathematical Problems in Engineering*, vol. 2012, pp. 1024 – 1036, May 2012.

[3] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 12, pp. 1518 – 1524, Dec. 1996.

[4] J. A. Bennell and J. F. Oliveria, “The geometry of nesting problems: A tutorial,” *European Journal of Operational Research,* vol. 184, no. 2, pp. 397 – 415, Jan. 2008.

[5] G. Wäscher, H. Haußner, and H. Schumann, "An improved typology of cutting and packing problems," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1109 – 1130, Dec. 2007.

[6] A. Bortfeldt and H. Gehring, “A hybrid genetic algorithm for the container loading problem,” *European Journal of Operational Research,* vol. 131, no. 1, pp. 143 – 161, May 2001.

[7] G. Scheithauer, “A three-dimensional bin-packing algorithm,” *Journal of Information Processing and Cybernetics,* vol. 27, no. 5 – 6, pp. 263 – 271, 1991.

[8] A. Lodi, S. Martello, and D. Vigo, "Models and bounds for two-dimensional level packing problems," *Journal of Combinatorial Optimization*, vol. 8, no. 3, pp. 363 – 379, Sept. 2004.

[9] G. Cornu´ejols, "Valid inequalities for mixed integer programs." *Mathematical Programming*, vol. 112, no. 1, pp. 3 – 44, Mar. 2008.

[10] L. A. Wolsey, "Technical note – facets and strong valid inequalities for integer programs." *Operations Research*, vol. 24, no. 2, pp. 367 – 372, Apr. 1976.

[11] M. Z. Gürbü, S. Akyokuş, İ. Emiroğlu, and A. Güran, "An efficient algorithm for 3D rectangular box packing," *Proceedings of Selected AAS 2009 Papers*, Skopje, Macedonia, Sept. 2009.

[12] L. Brunetta and P. Grégoire, "A general purpose algorithm for three-dimensional packing," *INFORMS Journal on Computing*, vol. 17, no. 3, pp. 328 – 338, Aug. 2005.

[13] K. K. Lai and J. W. M. Chan, "Developing a simulated annealing algorithm for the cutting stock problem," *Computers and Industrial Engineering*, vol. 32, no. 1, pp. 115 – 127, Jan. 1997.

[14] Y. G. Stoyan, "Mathematical methods for geometric design," *Advances in CAD/CAM: Proceedings of PROLAMAT82*, Leningrad, USSR, May 1982.

[15] 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*, vol. 23, no. 1 – 2, pp. 187 – 213, Jan. – Mar. 2016.

[16] K. Shiangjen, J. Chaijaruwanich, W. Srisujjalertwaja, P. Unachak, and S. Somhom, "An iterative bidirectional heuristic placement algorithm for solving the two-dimensional

knapsack packing problem," *Engineering Optimization*, vol. 50, no. 2, pp. 347 – 365, Mar. 2017.

[17] L. H. Cherri, L. R. Mundim, M. Andretta, F. Toledo, J.F. Oliveira, and M. A. Carravilla, "Robust mixed-integer linear programming models for the irregular strip packing problem," *European Journal of Operational Research*, vol. 253, no. 3, pp. 570 – 583, Sept. 2016.

[18] Y. Cui, Y. Yao, and Y. P. Cui, "Hybrid approach for the two-dimensional bin packing problem with two-staged patterns," *International Transactions in Operational Research*, vol. 23, no. 3, pp. 539 – 549, May 2016.

[19] I. Moon and T. V. L. Nguyen, "Container packing problem with balance constraints," *OR Spectrum*, vol. 36, no. 4, pp. 837 – 878, Oct. 2014.

[20] M. Hifi and R. M'Hallah, "A hybrid algorithm for the two-dimensional layout problem: the cases of regular and irregular shapes," *International Transactions in Operational Research*, vol. 10, no. 3, pp. 195 – 216, June 2003.

[21] A. Bortfeldt and D. Mack, "A heuristic for the three-dimensional strip packing problem," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1267 – 1279, Dec. 2007.

[22] L. Wei, W. Oon, W. Zhu, and A. Lim, "A reference length approach for the 3D strip packing problem," *European Journal of Operational Research*, vol. 220, no. 1, pp. 37 – 47, July 2012.

[23] W. Zhu and A. Lim, "A new iterative-doubling Greedy–Lookahead algorithm for the single container loading problem," *European Journal of Operational Research*, vol. 222, no. 3, pp. 408 – 417, Nov. 2012.

[24] Y. Juoung and S. D. Noh, “Intelligent 3D packing using a grouping algorithm for automotive container engineering,” *Journal of Computational Design and Engineering*, vol. 1, no. 2, pp. 140 – 151, April 2014.

[25] D. Whitley, “A genetic algorithm tutorial,” *Statistics and Computing*, vol. 4, no. 2, pp. 65 – 85, June 1994.

[26] M. Hifi and R. M’Hallah, “A literature review on circle and sphere packing problems: Models and methodologies,” *Advances in Operations Research*, vol. 2009, April 2009.

[27] K. J. Nurmela and P. R. J. Östergård, “Packing up to 50 equal circles in a square,” *Discrete & Computational Geometry*, vol. 18, no. 1, pp. 111–120, July 1997.

[28] Y. G. Stoyan and G. Yaskov, “Packing identical spheres into a rectangular parallelepiped,” *Intelligent Decision Support*, pp. 47–67.

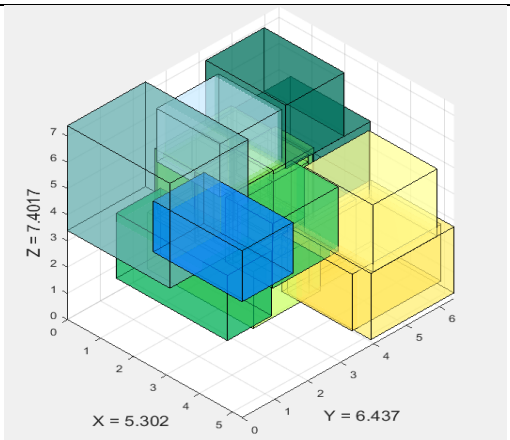
[29] Y. G. Stoyan and G. N. Yaskov, “Packing identical spheres into a right circular cylinder,” *Proceedings of the 5th ESICUP Meeting*, L’Aquila, Italy, April 2008

[30] Y. G. Stoyan, G. N. Yaskov, and G. Scheithauer, “Packing of various radii solid spheres into a paralleliped,” *Central European Journal of Operational Research*, vol. 11, no. 4, pp. 389–407, Dec. 2003.

[31] M. Sipser, *Introduction to the Theory of Computation, Second Edition*, Thomson Course Technology, Boston, Massachusetts, 2006.

[32] MathWorks, *Matlab Primer* (r2016b), Retrieved August 15th, 2017 from https://www.mathworks.com/help/releases/R2016b/pdf_doc/matlab/, 2016.

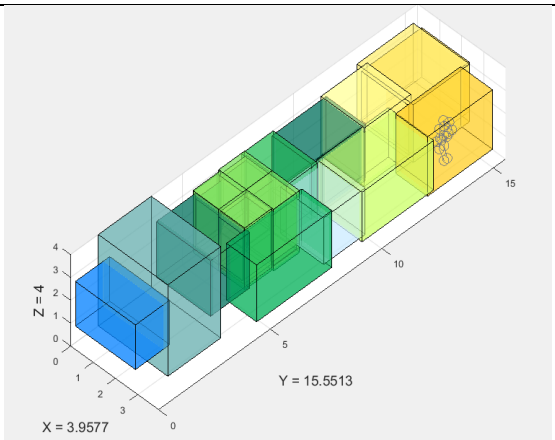
[33] Gurobi Optimization, Inc, *Gurobi Optimizer Quick Start Guide* (7.5), Retrieved August 15th, 2017 from http://www.gurobi.com/documentation/7.5/, 2017.



(a) overz_cuboid

(X, Y, Z): (5.302, 6.437, 7.402)

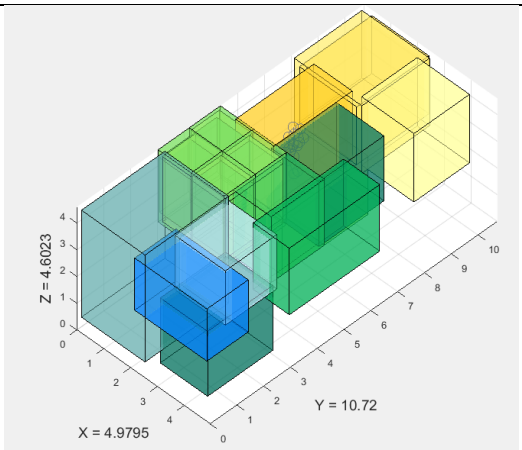
Container volume: 252.623



(b) overz_horizontal_4

(X, Y, Z): (3.958, 15.551, 4)

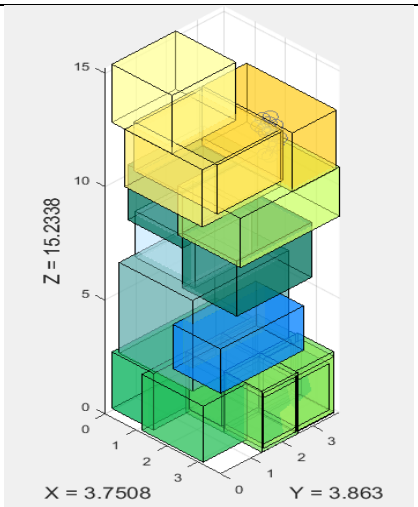
Container volume: 246.203



(c) overz_horizontal_5

(X, Y, Z): (4.980, 10.72, 4.602)

Container volume: 245.680

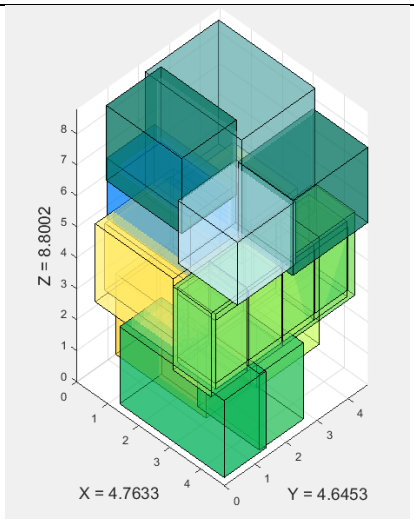


(d) overz_vertical_4

(X, Y, Z): (3.751, 3.863, 15.234)

Container volume: 220.742

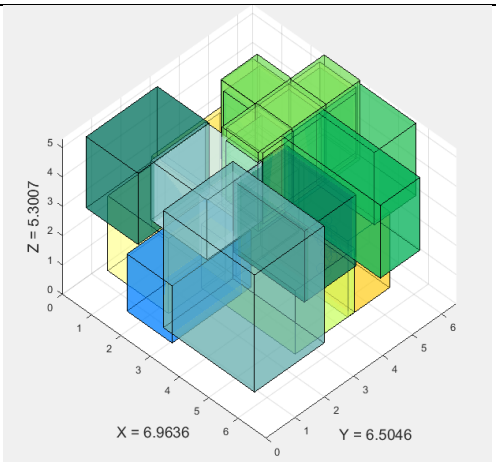
Figure A1: 10 layout designs for 14 task volumes using enhancements



(e) overz_vertical_5

(X, Y, Z): (4.763, 4.645, 8.800)

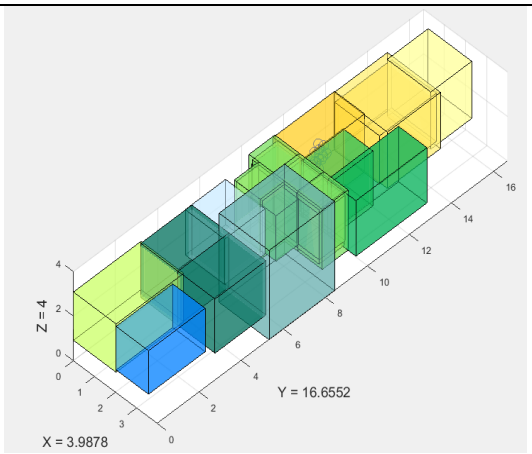
Container volume: 194.692



(f) solidz_cuboid

(X, Y, Z): (6.964, 6.505, 5.301)

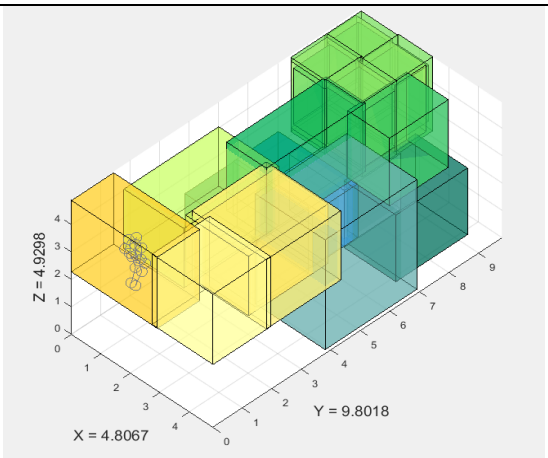
Container volume: 240.140



(g) solidz_horizontal_4

(X, Y, Z): (3.9878, 16.6552, 4)

Container volume: 265.670

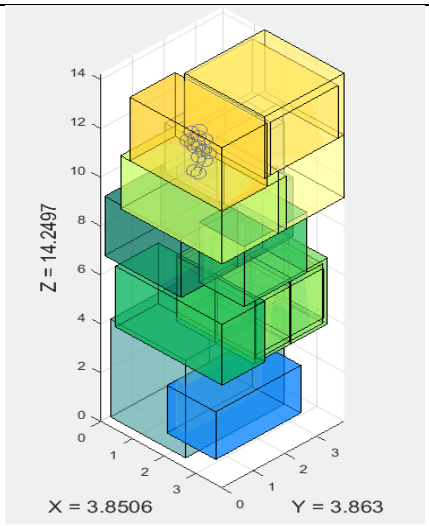


(h) solidz_horizontal_5

(X, Y, Z): (4.807, 9.802, 4.9298)

Container volume: 232.283

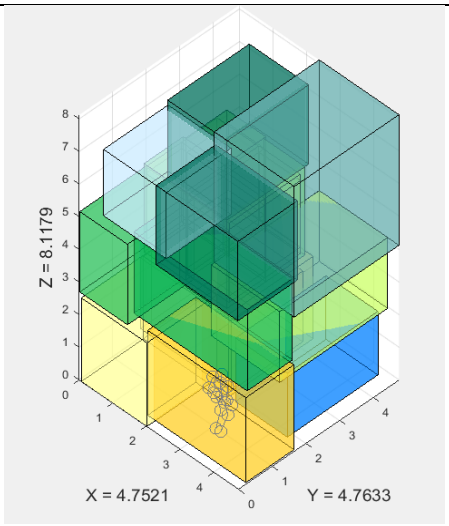
Figure A1, continued



(i) solidz_vertical_4

(X, Y, Z): (3.851, 3.863, 14.250)

Container volume: 211.989



(j) solidz_vertical_5

(X, Y, Z): (4.752, 4.763, 10.118)

Container volume: 229.009

Figure A1, continued

Table A2: Randomly generated cuboid dimensions for each scenario

	Exercise			Hygiene			Waste Collection & Management			Sleep			Hatch		
Scenario	Width	Length	Height	Width	Length	Height	Width	Length	Height	Width	Length	Height	Width	Length	Height
1	2.92	1.43	2.60	1.42	1.99	2.46	3.37	1.35	2.49	1.16	1.23	2.70	2.69	1.54	1.95
2	2.93	1.56	2.40	1.46	1.65	2.80	3.00	1.45	2.77	0.99	1.45	2.41	2.69	1.54	1.95
3	3.36	1.26	2.69	1.16	1.69	2.57	3.25	1.38	2.53	1.06	1.43	2.86	2.69	1.54	1.95
4	2.89	1.24	2.72	1.66	2.27	2.21	3.03	1.13	2.83	1.00	1.11	3.06	2.69	1.54	1.95
5	2.34	1.43	2.50	1.28	1.71	2.44	3.41	1.33	2.44	1.03	1.17	2.52	2.69	1.54	1.95
6	2.38	1.23	2.41	1.16	1.63	2.33	3.03	1.22	2.19	1.02	1.07	2.79	2.69	1.54	1.95
7	2.93	1.20	3.00	1.34	1.75	2.69	3.78	1.46	2.47	1.39	1.32	2.97	2.69	1.54	1.95
8	3.28	1.19	2.11	1.27	2.37	2.07	2.72	1.23	2.39	1.08	1.43	2.19	2.69	1.54	1.95
9	3.36	1.35	2.64	1.15	2.00	2.63	3.68	1.48	2.96	1.31	1.30	2.22	2.69	1.54	1.95
10	2.55	1.22	2.93	1.48	1.63	2.41	3.24	1.60	2.68	1.16	1.09	3.23	2.69	1.54	1.95
11	3.13	1.47	2.63	1.54	2.18	2.06	3.52	1.27	2.75	1.33	1.30	3.13	2.69	1.54	1.95
12	2.72	1.56	2.17	1.50	2.21	2.72	3.55	1.24	2.71	1.03	1.30	2.22	2.69	1.54	1.95
13	3.33	1.21	2.40	1.52	1.84	2.65	3.16	1.09	2.90	1.35	1.06	2.37	2.69	1.54	1.95
14	2.69	1.69	2.25	1.40	2.33	2.54	2.89	1.22	2.92	1.08	1.09	2.48	2.69	1.54	1.95
15	2.67	1.18	3.04	1.70	2.09	2.66	3.49	1.32	2.67	1.14	1.47	2.67	2.69	1.54	1.95
16	3.14	1.55	2.81	1.21	1.88	2.19	3.57	1.36	2.13	1.35	1.28	2.33	2.69	1.54	1.95
17	3.30	1.69	2.84	1.69	2.36	2.19	2.91	1.25	2.11	1.18	1.13	3.18	2.69	1.54	1.95
18	2.98	1.54	3.09	1.63	2.31	2.71	4.02	1.37	2.53	1.15	1.33	2.95	2.69	1.54	1.95
19	2.43	1.45	2.99	1.25	2.22	2.04	2.72	1.15	2.37	1.24	1.01	3.00	2.69	1.54	1.95
20	3.16	1.27	2.13	1.63	1.74	2.03	3.37	1.53	2.60	1.36	1.14	2.74	2.69	1.54	1.95