KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Technology

Algorithms for solving 3D cutting and packing problems with real-world constraints

Everton Fernandes da Silva

Supervisors:
Prof. dr. T. Wauters
Prof. dr. G. Vanden Berghe

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Technology (PhD)

June 2020

Algorithms for solving 3D cutting and packing problems with real-world constraints

Everton FERNANDES DA SILVA

Examination committee:
Prof. dr. F. Desplentere, chair
Prof. dr. T. Wauters, supervisor
Prof. dr. G. Vanden Berghe, supervisor
Prof. dr. K. Aerts
Prof. dr. D. Cattrysse
Prof. dr. J. Belien
Prof. dr. A. Martinez-Sykora
(University of Southampton, United Kingdom)

dr. K. Dockx
(OMP, Belgium)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Technology (PhD)

June 2020

© 2020 KU Leuven – Faculty of Engineering Technology
Uitgegeven in eigen beheer, Everton Fernandes da Silva, Celestijnenlaan 200A box 2402, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Acknowledgements

First of and foremost, I would like to thank Greet for giving me the opportunity of doing my PhD in a well-known institution such as this research group and, of course, Tony for agreeing to supervise me throughout these past four years. I am also thankful for all the past and current members of CODeS for always being welcoming. A special thanks to Túlio and Luke, who since day one helped me with my adaptation and provided companionship through all these years living far from home. Their conversations and support will not be forgotten and I will take these friendships with me into my future.

Of course, I have to mention Luke, Manos, Michiel, Túlio, and, more recently, Hatice (the gang in the back). I will always remember our jokes, serious conversations and not-so-serious conversations, especially those after working hours in our usual bar. When you live alone and spend the majority of your time in an office, people may not realize it, but these close friends become your "local family". So thank you for providing all the nice moments I will keep with me. I would also like to take this opportunity to thank other people that during this period of my life and who all, in one way or another, made an impact upon me: Adriana, Danilo, Diogo, Gustavo, Franklina, Larissa Tebaldi, Larissa Lotufo, Luiz Henrique and Priscila.

Leaving your home to live in a faraway country with a different way of life for four years is not an easy task. For that I would like to thank my family for supporting and encouraging me to embrace this challenge.

Finally, and most importantly, I would like to thank Flávia for always being by my side despite the distance. Without her patience, wisdom and love I would definitely not have been able to finish this PhD.

Abstract

Three-dimensional cutting and packing problems consist of a set of items which must be cut from or placed inside one or more larger items. Such problems enforce non-overlapping constraints to ensure items completely fit inside their respective larger items. Besides our daily activities, these problems are largely found in the logistics and manufacturing industry. Such problems tend to be intuitively solved by human planners who are able to visualize and deploy general rules of thumb to arrive at high quality solutions. Replicating this human behavior in computational algorithms is exceedingly difficult. But it is necessary to do so if one wishes to solve large-scale real-world instances of such problems which even skilled human planners cannot handle. Therefore, the objective of this thesis is to provide efficient solution approaches for realistic 3D cutting and packing problems with multiple real-world constraints. Beginning with regular shapes and more straightforward objectives, this thesis will progress toward more extreme irregular cases.

As with all research domains, before addressing regular 3D packing problems it is necessary to conduct a thorough review of existing methods for solving such problems. Given the existence of heuristic surveys and the absence of comparisons concerning exact solution methods, we reimplement the most notable exact approaches from the literature. We also generate new instances in order to properly evaluate the positive and negative aspects of each method in relation to one another when they are all executed on up-to-date hardware. After considering the problem variants analyzed in this review, we then address the container loading problem with real-world constraints. The main objective is to develop a solution approach which is flexible enough to provide good quality solutions across different subsets of constraints. After experimenting with heuristic methods (which currently produce the best results) and exact approaches (which are more flexible in terms of adaptation), we arrived at a well-performing combination of both approaches: a matheuristic which is structured in such a way that it can be easily adapted for other problem variants. We also updated instances from the literature so that they become more representative

of real-world conditions.

This thesis ends by addressing one of the more extreme cases of 3D cutting and packing problems: the extraction of objects from regular and irregular 3D materials. This problem is directly related to many different real-world applications which are currently studied in isolation. Our objective is to efficiently address the problem in a general way that can be used across all such applications. Therefore, we propose a matheuristic approach capable of quickly generating high quality solutions. We also consider linear and non-linear mathematical formulations to evaluate our solution method. Experiments were conducted with many different object and material shapes in order to evaluate the performance of our solution methods across a wide range of scenarios and to establish benchmark instances for future research.

This thesis was able to address different 3D cutting and packing problems with different sets of real-world constraints and was able to adapt to these different problems providing robust solution approaches: fast and high quality solutions in both best- and worst-case scenarios. This work also provides the foundations for further research involving realistic 3D packing problems.

Beknopte samenvatting

Driedimensionale *cutting & packing* is de discipline die probeert een verzameling voorwerpen uit een groter voorwerp te snijden of in een groter voorwerp te plaatsen. Enkele randvoorwaarden liggen voor de hand: de voorwerpen mogen niet overlappen en moeten bovendien volledig binnen het grote volume passen. Dit probleem uit het dagelijks leven komt ook veelvuldig voor in de logistiek en de productie-industrie.

Menselijke planners beschikken over intuïtieve oplossingsvaardigheden voor dergelijke problemen. Ze slagen erin om door middel van visualisaties en vuistregels redelijk goede resultaten te bekomen voor cutting & packing problemen. Het blijkt echter bijzonder moeilijk om deze menselijke handelswijzen te vertalen naar een algoritmische aanpak. Beslissingsondersteuning is nochtans onontbeerlijk wanneer we oplossingen ambiëren voor grootschalige reële probleeminstanties die zelfs voor getalenteerde menselijke planners veel te uitdagend zijn. Daarom heeft deze thesis als doelstelling om efficiënte oplossingsmethoden te ontwikkelen voor 3D cutting & packing problemen met realistische randvoorwaarden. Na het bestuderen van eenvoudige vormen en objectieven, evolueert deze thesis naar het optimaliseren van extreem onregelmatige vormen.

Vooraleer de 3D problemen aan te pakken, geven we een grondig overzicht van oplossingsmethoden voor cutting & packing. Er bestaan al overzichtspublicaties over heuristieken maar nog geen vergelijkende studies over exacte oplossingsmethoden. Daarom herimplementeren we de meest prominente exacte methoden uit de wetenschappelijke literatuur. We genereren ook nieuwe probleeminstanties om de positieve en negatieve aspecten van deze methoden te vergelijken wanneer uitgevoerd op hedendaagse hardware.

Na een analyse van bestaande probleemvarianten voor *container loading*, modelleren we realistische randvoorwaarden voor dit probleem. De voornaamste doelstelling is het ontwikkelen van flexibele oplossingsmethoden die goed func-

tioneren voor problemen met verschillende combinaties van deze beperkingen. Experimenten met heuristieken (die momenteel het best werken) en exacte methoden (die het eenvoudigst aan te passen zijn), leiden naar een performante combinatie van de beide benaderingen: een goed gestructureerde *matheuristiek*. We hebben de instanties van de literatuur zodanig uitgebreid dat ze overeen komen met reële gevallen.

Deze thesis eindigt met het oplossen van de meest extreme 3D cutting & packing problemen: het extraheren van objecten uit regelmatige en onregelmatige 3D-volumes. Dit probleem komt overeen met verschillende reële toepassingen, waarvoor het onderzoek momenteel in afzonderlijke disciplines gebeurt. Ons doel is om een efficënte oplossingsmethode te ontwikkelen, die algemeen toepasbaar is in de verschillende domeinen. We stellen opnieuw een matheuristiek voor, en beschouwen zowel lineaire als non-lineaire formuleringen om het nieuwe algoritme te evalueren. Voor de experimenten onderzoeken we verschillende vormen van de objecten en containervolumes en een brede waaier aan scenario's. Het experimenteel werk resulteert in nieuwe maatstaven voor verder onderzoek.

Deze thesis is erin geslaagd om verschillende 3D cutting & packing problemen met diverse realistische randvoorwaarden op te lossen door middel van nieuwe robuuste algoritmen. Het resultaat is een snelle methode die goede oplossingen berekent in zowel ideale als *worst case* omstandigheden. Dit werk legt ook een basis voor verder onderzoek gerelateerd aan realistische 3D cutting & packing problemen.

Abbreviations

2D Two-dimensional. 73

3D Three-dimensional. iii–vi, 5–8, 10, 11, 13, 14, 19, 35–37, 39–41, 67, 69–71, 73, 74, 90, 95–97

CLP Container Loading Problem. 40

DFS Depth-First Search. 16, 25

EMOC Extracting the Maximal Object from a Convex Material. xvi, 74–76, 82, 84, 85, 89

EMOG Extracting the Maximal Object from a General Material. xvi, 84, 85, 92

EMOG-FO Extracting the Maximal Object from a General Material with Fixed Orientation. xvi, 85, 89

IIPP Identical Item Packing Problem. 11, 16, 35

ILS Iterated Local Search. 76, 79

LP Linear programming. 70, 76, 77, 89

MBSBPP Multiple Bin Size Bin Packing Problem. 11, 14, 35

MILP Mixed integer linear programming. xvi, 10, 11, 15, 24, 70, 84, 87, 89, 90

MSSCSP Multiple Stock Size Cutting Stock Problem. 11

NLP Non-linear programming. xiv, xvi, 70, 74, 75, 77, 78, 81–83, 89

ODP Open Dimension Problem. 11, 35

RBPP Residual Bin Packing Problem. 11

RCSP Residual Cutting Stock Problem. 11

SBSBPP Single Bin Size Bin Packing Problem. 11

SKP Single Knapsack Problem. 9, 11, 16, 18, 22, 24, 37, 41

SLOOM Single Large Object Output Maximization. xv, 18–20

SLOPP Single Large Object Placement Problem. 9, 11, 16, 18, 22, 35, 37, 40–42, 45, 47, 51, 56, 63, 67

SSSCSP Single Stock Size Cutting Stock Problem. xvi, 11, 41, 42, 63, 64, 66, 99, 100

Contents

Abstract iii

Beknopte samenvatting v

Abbreviations viii

Contents ix

List of Figures xiii

List of Tables xv

1 Introduction 1

2 A comparison of exact methods for single container problems 7

2.1 Introduction 8

2.2 Exact methods 10

2.2.1 Mathematical models 10

2.2.2 Combinatorial branch and bound 16

2.3 Problem adaptations 18

2.3.1 Single Large Object Output Maximization adaptation . 18

2.4 Computational results 19

2.4.1 Instances 20
2.4.2 Single Large Object Output Maximization 23
2.4.3 Generated Instances 26
2.4.4 Rotations 32
2.4.5 Discussion 34
2.5 Conclusions 36

3 A matheuristic framework for the three-dimensional Single Large Object Placement Problem with practical constraints 39

3.1 Introduction 40
3.2 Practical constraints 42
3.3 Solution method 43
3.3.1 Wall generation 44
3.3.2 Mathematical formulation 45
3.3.3 Improvement phase 51
3.4 Computational Results 51
3.4.1 Sensitivity Analysis 53
3.4.2 Test case 1 55
3.4.3 Test case 2 56
3.4.4 All test cases 57
3.4.5 Results discussion 60
3.5 Multiple Container Approach 62
3.6 Conclusions 67

4 Extracting maximal objects from three-dimensional solid materials 69

4.1 Introduction 70
4.2 Related work 71
4.2.1 Design centering 71

4.2.2 Diamond and gemstone cutting 72
4.2.3 Irregular object cutting and packing in 3D Euclidean space 73
4.3 Convex outer object 73
4.3.1 Non-linear programming formulation 74
4.3.2 Matheuristic approach for the EMOC 75
4.3.3 Computational study 77
4.4 Non-convex solid material 84
4.4.1 Matheuristic adaptation for the EMOG 85
4.4.2 MILP formulation 87
4.4.3 Computational study 89
4.5 Conclusion 93

5 Conclusion **95**

A SSSCSP formulations **99**

B EMOC/EMOG instances **103**

Bibliography **107**

List of publications **113**

List of Figures

1.1 1D, 2D and 3D cutting and packing problem examples. 2

1.2 Cutting and packing subproblems (Wäscher et al. 2007). 4

2.1 Relative positions for placing box k adjacent to box i. 14

2.2 Corner points generation 17

2.3 SSA60 and SSA70 instances. 26

2.4 SSA80 and SSA90 instances. 27

2.5 SSB60 and SSB70 instances. 28

2.6 SSB80 and SSB90 instances. 29

2.7 SSC60 and SSC70 instances. 30

2.8 SSC80 and SSC90 instances. 31

2.9 Orientation comparison for the SSA60, SSA70 and SSA80 instances. 33

3.1 Structure of the solution method. 43

3.2 Complex block generation. 44

3.3 Truck measurements. 47

3.4 Impact of the wall quality parameter for test case 1. 54

3.5 Impact of the wall quality parameter for test case 5. 54

3.6 Impact of the number of extra walls parameter for test case 2. . 55

3.7 Impact of the number of extra walls parameter for test case 5. . 55

3.8 Average improvement percentage points from first to best solution found. 59

3.9 Average number of iterations. 60

3.10 Time spent in each module of the solution approach (part I). . . 61

3.11 Time spent in each module of the solution approach (part II). . 62

4.1 The two problem variants addressed in this chapter. 74

4.2 The helmets extracted from a *frustum*-shaped material (matheuristic x best NLP solution). 83

4.3 The bananas extracted from a *convex stone*-shaped material (matheuristic x best NLP solution). 83

4.4 Instances ordered by average execution time of the matheuristic approach. 84

4.5 Convex hull of a *star*-shaped material. 85

4.6 Updated convex hull by the addition of cutting planes. 86

4.7 Tetrahedra composing a *banana*-shaped material. 87

B.1 Solid materials and objects (part 1). 103

B.2 Solid materials and objects (part 2). 104

List of Tables

1.1 Thesis structure. 6

2.1 MILP formulations investigated. 11
2.2 Input data. 11
2.3 Decision variables. 12
2.4 Input minimization model characteristics. 15
2.5 SLOOM adaptations. 20
2.6 Characteristics of the considered instance sets. 22
2.7 CHEN, TSAI and TSAIRW instances. 23
2.8 SA instances (optimal solutions). 24
2.9 SA instances (non-optimal solutions). 24
2.10 LN instances. 25
2.11 BR instances. 25
2.12 Size of the adapted formulations. 35

3.1 Input data and variables. 46
3.2 Parameters. 48
3.3 Variables. 48
3.4 Constraints considered in each set of experiments. 53

3.5 Orientation and stability constraints. 56
3.6 Container parameters. 57
3.7 Orientation, stability, weight limit and load balance constraints. 57
3.8 Average container occupation for different combinations of constraints. 58
3.9 Number of containers used in the solution for each SSSCSP instance. 66

4.1 Input data 74
4.2 Decision variables 75
4.3 Comparison of the different strategies used in the *perturb* procedure. 80
4.4 NLP and matheuristic results when solving the EMOC. 82
4.5 Parameters 88
4.6 Additional decision variables 88
4.7 MILP model compared to matheuristic approach for the EMOG-FO. 89
4.8 Matheuristic performance for the EMOG with instances containing non-convex solid materials. 92

B.1 Number of vertices and faces of each item. 105

Chapter 1

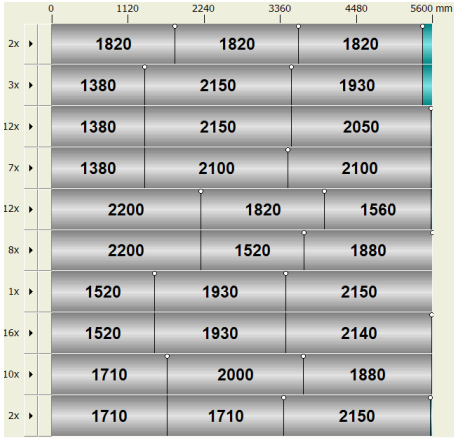
Introduction

Cutting and packing problems involve defining the position of a (sub)set of smaller items inside a set of larger items (containers) in accordance with a predefined objective function while ensuring that those smaller items do not overlap one another. These problems are found either in our everyday life and in the industry. For example, when a person goes to the supermarket they must deal with a linked list of various packing problems: placing groceries into the cart, placing them into a set of bags at the checkout, placing those bags into their car boot and finally placing those goods into their presses at home. In these examples the objectives might be to pack as many groceries into as few bags as possible while ensuring that each product's demand is met and preserving the integrity of those products by not, say, placing heavy items atop lighter ones. What is true in everyday life is also true for industry and academia: cutting and packing problems are everywhere.

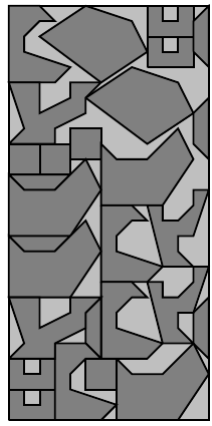
Depending on the number of spatial dimensions considered, cutting and packing problems have different applications (Figure 1.1). From the many industrial processes in which the one-dimensional version of this problem arises, the most common are related to generating patterns to cut objects such as steel bars, rolls of paper or aluminium. Meanwhile, the most common problem variant addressed in the literature involves minimizing the number of containers (bars or rolls, for example) required to cut all the demanded items. Cherri et al. (2014) provide a complete survey of exact and heuristic solution methods for such problems.

When considering two dimensions, a wide range of new applications can be found such as cloth, leather, glass and metal cutting. Given the shapes of the objects present in real-world two-dimensional applications, irregular problem

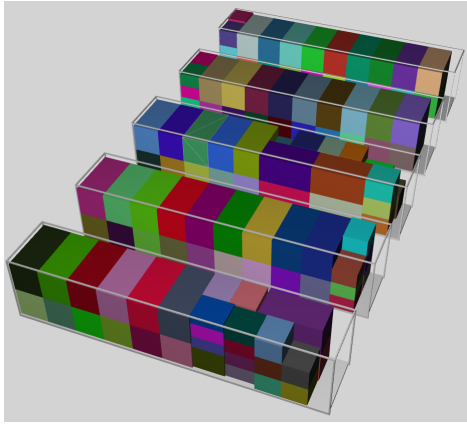
Figure 1.1: 1D, 2D and 3D cutting and packing problem examples.



(a) One-dimensional.



(b) Two-dimensional.



(c) Three-dimensional.

variants are the majority in nature. Irregular problems correspond to those in which either the small or the large item have irregular shapes (their sides and interior angles are not all the same). For example pieces of clothes or shoes and metallic car parts rarely have a regular shape. For these problems, Leao et al. (2019) provide a review of mathematical models proposed thus far, while Alvarez-Valdes et al. (2018) have reviewed existing heuristic approaches.

Finally, when considering the three-dimensional case, a generalization of the previous problems, one encounters even more applications in real-world environments such as in the logistics, manufacturing, furniture, and gemstone industries. Given the difficulty in solving the problem when considering all three dimensions, most research is dedicated to regular cutting and packing problems. The most common application addressed concerns container loading and transportation. Bortfeldt and Wäscher (2013) provide an overview of practical constraints addressed in the literature, while Zhao et al. (2016) present a survey on published heuristic methods for this problem.

Depending on the specific variant of the cutting and packing problem being addressed, items can be regular or irregular, while the objective function might concern maximizing the value placed or minimizing the number of containers used. Wäscher et al. (2007) proposed a useful typology through which to define each problem variant.

Figure 1.2 illustrates how Wäscher et al. (2007) subdivide cutting and packing problems into three main groups based on their objective functions: Input Minimization, Output Maximization and Open Dimension. Problems in the Input Minimization group seek to minimize the number of containers used to place all items, while for Output Maximization problems the goal is to maximize the value (volume, for example) of the items placed. Finally, the Open Dimension problem is a generalization of Input Minimization that consists of a single container for which one, two or all three dimensions are not fixed. In this case, the objective is to place all the items while minimizing the open dimension(s).

Within each of these three groups, an additional subdivision exists based on how heterogeneous the containers and items are. In summary, the container(s) used may be identical, weakly heterogeneous or strongly heterogeneous. The small items can also be either identical, weakly heterogeneous or strongly heterogeneous. According to Wäscher et al. (2007), weakly heterogeneous items are grouped together into relatively few groups according to the total number of items. These groups are composed of items which are identical with respect to shape and size. By contrast, strongly heterogeneous items have very few items of identical shape and size. The various combinations of container and item heterogeneity is what results in each of the subproblems listed in Figure 1.2.

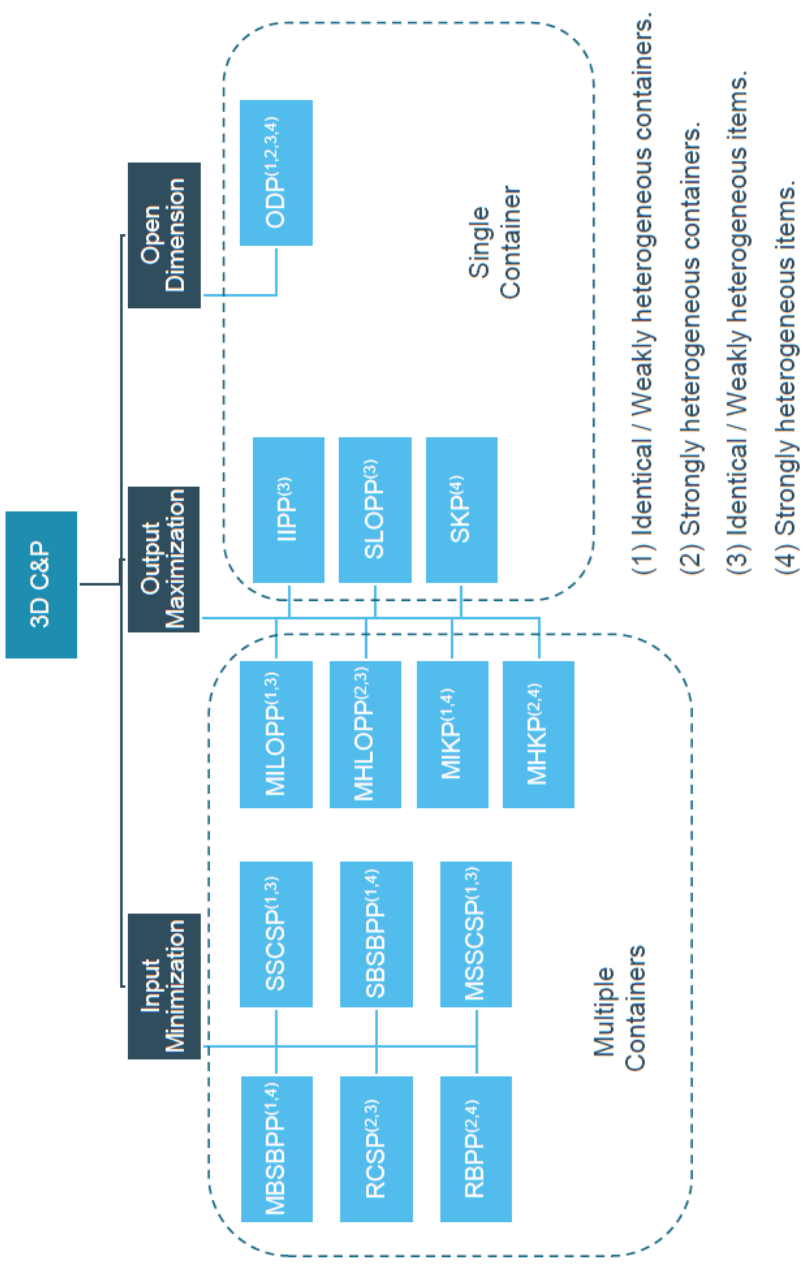


Figure 1.2: Cutting and packing subproblems (Wäscher et al. 2007).

This thesis addresses the 3D cutting and packing problem, the most difficult of them and also not properly well addressed for real-world problems given its very recent, limited and problem-specific literature. One interesting aspect of this problem is that it differs from most other well-known combinatorial optimization problems given its visual nature. This characteristic enables the problem to be generally well-solved using human intuition, which develops after a lot of practice and training from working on the same problem for an extended period of time.

Despite this expertise, for some difficult cases it may take weeks for a good solution to be found. A good example of this phenomenon is a jigsaw puzzle, a packing problem with a single optimal solution. A jigsaw of only a few pieces is very easy to solve, but the few seconds required to solve a jigsaw of 20 pieces does not scale linearly. A 2000 piece jigsaw would probably take a week or longer to solve.

One of the primary goals of Operations Research consists of bridging the gap between theory and practice. The implication being that most approaches inhabit the realm of idealized circumstances and situations which would not arise in practice or which do not incorporate necesssary real-world constraints. We address 3D cutting and packing problems: from regular boxes and containers with straightforward objectives to more extreme irregular cases. To solve these problems we will have to overcome the difficulties experienced by human decision makers. This means solving these problems in a robust and efficient way.

This thesis begins by considering the regular case and providing an experimental survey of exact approaches for solving 3D packing problems (Chapter 2). The motivation for this survey is that while there are well-known exhaustive comparisons concerning heuristic approaches, there is no such comparison for exact methods. We will therefore not only reimplement the most significant exact approaches and compare their performance against existing literature instances, but also create new challenging instances to properly evaluate the pros and cons of each approach. The idea is to evaluate what tactics already present in existing methods can be re-used and which can be improved for use in new solution methods.

With the knowledge acquired from this experimental survey, we can now properly address the 3D packing problem with real-world constraints (Chapter 3). Two objectives are considered when solving the problem: to address a realistic scenario and provide a solution approach flexible enough to adapt itself for different scenarios. After analyzing the current best performing heuristic methods and exact approaches we achieve our objectives by combining the best of both into a matheuristic approach to solve a specific problem variant considering different combinations of real-world constraints. We compare it

with the state-of-the-art heuristics and also demonstrate how it can be easily adapted to other problem variants. The instances which reflect realistic problem characteristics have been published online to encourage further reasearch on this topic.

After providing some academic foundations for realistic regular 3D packing, this thesis ends by investigating a more difficult case: extracting maximal objects from solid materials (Chapter 4). Given that this problem is related to very different real-world applications, we executed experiments with the most different object and material shapes in order to evaluate the performance of our solution methods in different scenarios. The instances used for the experiments have also been made publicly available online in order to provide benchmarks for future research.

Table 1.1: Thesis structure.

3D Cutting & Packing	Chapter 1	Chapter 2	Chapter 3	Chapter 4	Chapter 5
Regular	Introduction	Review	Matheuristic	-	Conclusions
Irregular		-	-	Matheuristic	

Table 1.1 summarizes what each chapter addresses. By providing insightful analyses, methodological components, instances and benchmarks, we hope that this thesis will incentivize more researchers to address realistic 3D cutting and packing problems. Not only to establish the area as popular as research area in theory, but also to push that theory into practice.

Chapter 2

A comparison of exact methods for single container problems

Although often studied in the literature, very few exact approaches have been developed for 3D packing problems. The vast majority of publications focus on developing heuristic methods, as is made evident by existing surveys. While heuristics are known to significantly outperform exact approaches, if there is no attempt to better understand existing exact approaches and what could be done to improve their efficiency then this gap will never close. Another reason to study exact approaches is to pinpoint their strengths and limitations. The answers to all these questions will likely define useful research directions for generating improved solution methods capable of solving a wide range of 3D packing problems.

Therefore, to fill this academic gap, we adopted a slightly unorthodox approach to our literature review. By adapting the most relevant exact methods to consider the same set of problems, we re-implemented and compared their performance on up-to-date computers when solving instances from the literature in addition to newly generated instances. The results from these experiments allow us to evaluate how each method performs across various scenarios.

This chapter is the result of one such review. Its content was published as: Silva, E. F., Toffolo, T. A. M. and Wauters, T. (2019). Exact methods for three-dimensional cutting and packing: A comparative study concerning single container problems. Computers and Operations Research, 109, 12-27. Túlio

Toffolo contributed by providing insights on how to compare the solution methods and what to analyze and Tony Wauters supervised this research.

2.1 Introduction

Three-dimensional packing problems are of crucial relevance for manufacturing and logistics industries and essentially consist of placing a set of small items inside a set of larger items. Considering the most common problem's characteristics addressed in the literature, the current chapter restricts its scope to problems in which items are rectangular shaped (cuboids). For notation purposes, small and larger items will henceforth be referred to as boxes and containers, respectively. The primary constraints associated with these problems are that (*i*) boxes must not overlap and (*ii*) boxes must be placed completely inside their respective containers. A secondary set of constraints concern real-world requirements and specific practical situations depending upon the characteristics of boxes and containers in addition to various safety measures which must be respected during transportation such as orientation, stability and weight limitations. Good quality placement of boxes (also known as packing patterns) inside a container should guarantee safety during their transportation, thereby improving general productivity.

Due to the complexity of these problems, most new published research in the literature proposes heuristic methods which are capable of quickly finding solutions of acceptable quality in terms of container utilization. Examples of heuristic methods were published by Parreño et al. (2008), He and Huang (2011), Gonçalves and Resende (2012), Lim et al. (2012), Araya and Riff (2014) and Toffolo et al. (2017). A thorough comparison between the heuristic methods, with the exception of Toffolo et al. (2017) which was published later, was compiled by Zhao et al. (2016). By contrast, only a few **exact approaches** exist in the literature. The most notable examples are the mathematical formulations proposed by Chen et al. (1995), Fasano (2004), Hifi et al. (2010), Junqueira et al. (2012), Tsai et al. (2015), Paquay et al. (2016) and the specialized Branch and Bound method proposed by Martello, Pisinger, and Vigo (2000).

While heuristic proposals have been comparatively assessed by Zhao et al. (2016), there is no current research comparing exact approaches for each variant of the 3D packing problem. Therefore, **the objective of the research reported in this chapter is not to propose new methodologies, but instead to provide an up-to-date comparative study concerning the most significant exact methods associated with 3D packing problems**.

This chapter provides a detailed comparison of such methods by adapting them in accordance with the problem being analyzed and the 3D packing problem typology defined by Wäscher et al. (2007) and evaluating their performance by means of running extensive experiments. Given that the idea is to compare the methods in the problem's most basic configuration the only additional constraints considered are the orientation constraints which allow more flexibility regarding the solution space of the problem compared against considering no rotations.

Since it is not feasible to evaluate all exact methods on all problem variants we limit ourselves to consider the Single Large Object Placement Problem (SLOPP) and the Single Knapsack Problem (SKP), variants of the Output Maximization Problem. Both problems consider a single container whereas the SLOPP considers weakly heterogeneous boxes and the SKP considers strongly heterogeneous boxes.

The reason for focusing on problems that consider only one container is that, besides their many relevant applications in the transport logistics and manufacturing industries, these are the basis for the other problems, with most solution methods that consider multiple containers decomposing the problem into multiple single-container problems. Mohanty et al. (1994), Martello, Pisinger, and Vigo (2000), Crainic et al. (2009) and Elhedhli et al. (2017) provided examples of methods that employ this kind of strategy.

The classic benchmark datasets proposed by Loh and Nee (1992) and Bischoff and Ratcliff (1995) which contain a variety of item configurations and box quantities ranging from 69 to 476 are employed insofar to enable a thorough comparative analysis concerning existing exact approaches. New instances derived from the packing generator proposed by E. Silva, Oliveira, et al. (2014) are also employed to provide information regarding scaling behavior and influence of the percentage of container occupation for each of the methods undergoing analysis. Results demonstrate how poorly these methods perform, being unable to solve instances with fewer than 20 boxes to optimality within 6h of runtime.

This chapter is organized as follows, Section 2.2 presents the considered exact approaches. A detailed description concerning the adaptation of the selected exact methods and instances employed is provided in Section 2.3. The results obtained (Section 2.4) provide further insight concerning the situations in which each method performs well and, similarly, the bottlenecks for each method. Conclusions which are crucial for the continued development of existing exact methods and generation of new formulations are presented in Section 2.5.

2.2 Exact methods

After a thorough investigation of the 3D packing literature, the most referenced exact methods are mathematical formulations employing mixed integer linear programming MILP solvers by Chen et al. (1995), Fasano (2004), Hifi et al. (2010), Junqueira et al. (2012), Tsai et al. (2015) and Paquay et al. (2016). The only exception is the specialized Branch and Bound method proposed by Martello, Pisinger, and Vigo (2000). A more detailed description concerning these different approaches is provided throughout the following sections.

2.2.1 Mathematical models

As mentioned before, it was decided that only the primary constraints (overlapping and container dimensions) that define the 3D packing problem together with the orientation constraints would be considered when performing experiments. Possible additional constraints proposed in the literature are not included in the implementations and are therefore not considered during the experiments. For the regular assignment of cuboids, orientation constraints allow boxes to be rotated only by 90 degrees on each axis. That way the box will always be placed with its sides parallel to each of the containers sides. The number of allowed orientation and which axis can be rotated depends on each instance set. The reason for including only these constraints is that rotations enable more possibilities for positioning boxes which, in turn, can generate solutions with greater value than compared to when no rotations are permitted. All the mathematical formulations detailed throughout this section only include the aformentioned constraints. Orientation constraints are specified only if the original formulation considered them; otherwise adapted orientation constraints are employed, which will be detailed later in Section 2.3.

The MILP models proposed by Chen et al. (1995), Hifi et al. (2010), Tsai et al. (2015) and Paquay et al. (2016) were originally proposed for Input Minimization Problems, each one considering specific problem characteristics. The remainder of the investigated models, introduced by Fasano (2004) and Junqueira et al. (2012), were originally proposed for Output Maximization Problems with different characteristics. Table 2.1 details for which problem each formulation was originally proposed, according to the typology of Wäscher et al. (2007).

This section first describes the model proposed by Chen et al. (1995) before subsequently discussing the differences between their formulation and the remaining ones. The reason for choosing the oldest formulation is that it was used as the basis for the majority of the other models.

Table 2.1: MILP formulations investigated.

Formulation	Input Minimization	Output Maximization
Chen et al. (1995)	MBSBPP | MSSCSP | RCSP | RBPP	
Hifi et al. (2010)	SBSBPP | SSSCSP	
Tsai et al. (2015)	ODP	
Paquay et al. (2016)	MBSBPP	
Fasano (2004)		IIPP | SLOPP | SKP
Junqueira et al. (2012)		IIPP | SLOPP | SKP

Input Minimization mathematical models

Chen et al. (1995) proposed a MILP model for the 3D Input Minimization Problem with rotations considered. Its primary characteristic is the consideration of containers with varying dimensions. The extra set of constraints considered explicitly in the model are the orientation constraints while other real-world constraints such as weight distribution and container load bearing are further discussed and explained on how to adapt them into the model. Note that Chen et al. (1995) do not specify whether the model was proposed for weakly or strongly heterogeneous boxes. The input data and variables employed within the formulation are detailed in Tables 2.2 and 2.3, respectively:

Table 2.2: Input data.

$\mathcal{I}$:	set of boxes
$\mathcal{J}$:	set of available containers
M :	an arbitrary large number
p_i, q_i, r_i :	length, width and height of box i, respectively
L_j, W_j, H_j :	length, width and height of container j, respectively

Table 2.3: Decision variables.

s_{ij} :	binary variable which is equal to 1 if box i is placed in container j and 0 otherwise
n_j :	binary variable which is equal to 1 if container j is used and 0 otherwise
x_i, y_i, z_i :	continuous variables indicating the x, y and z coordinates of the front-left-bottom corner of box i, respectively
l_{xi}, l_{yi}, l_{zi} :	binary variables indicating whether the length of box i is parallel to the X-, Y-, or Z-axis, respectively
w_{xi}, w_{yi}, w_{zi} :	binary variables indicating whether the width of box i is parallel to the X-, Y-, or Z-axis, respectively
h_{xi}, h_{yi}, h_{zi} :	binary variables indicating whether the height of box i is parallel to the X-, Y-, or Z-axis, respectively
$a_{ik}, b_{ik}, c_{ik},$ d_{ik}, e_{ik}, f_{ik} :	binary variables indicating the relative position of boxes i and k (left, right, behind, in front, below, and above, respectively)

The problem formulation is presented by Equations (2.1)-(2.26):

$$\min \quad \sum_{j\in\mathcal{J}} L_j W_j H_j n_j - \sum_{i\in\mathcal{I}} p_i q_i r_i \tag{2.1}$$

$$\text{s.t.} \quad x_i + p_i l_{xi} + q_i w_{xi} + r_i h_{xi} \le x_k + (1 - a_{ik})M, \qquad \forall\, i,k \in \mathcal{I}, i < k, \tag{2.2}$$

$$x_k + p_k l_{xk} + q_k w_{xk} + r_k h_{xk} \le x_i + (1 - b_{ik})M, \qquad \forall\, i,k \in \mathcal{I}, i < k, \tag{2.3}$$

$$y_i + q_i w_{yi} + p_i l_{yi} + r_i h_{yi} \le y_k + (1 - c_{ik})M, \qquad \forall\, i,k \in \mathcal{I}, i < k, \tag{2.4}$$

$$y_k + q_k w_{yk} + p_k l_{yk} + r_k h_{yk} \le y_i + (1 - d_{ik})M, \qquad \forall\, i,k \in \mathcal{I}, i < k, \tag{2.5}$$

$$z_i + r_i h_{zi} + q_i w_{zi} + p_i l_{zi} \le z_k + (1 - e_{ik})M, \qquad \forall\, i,k \in \mathcal{I}, i < k, \tag{2.6}$$

$$z_k + r_k h_{zk} + q_k w_{zk} + p_k l_{zk} \le z_i + (1 - f_{ik})M, \qquad \forall\, i,k \in \mathcal{I}, i < k, \tag{2.7}$$

$$a_{ik} + b_{ik} + c_{ik} + d_{ik} + e_{ik} + f_{ik} \ge s_{ij} + s_{kj} - 1, \qquad \forall\, i,k \in \mathcal{I}, j \in \mathcal{J}, i < k, \tag{2.8}$$

$$\sum_{j\in\mathcal{J}} s_{ij} = 1, \qquad \forall\, i \in \mathcal{I}, \tag{2.9}$$

$$\sum_{i\in\mathcal{I}} s_{ij} \le M n_j, \qquad \forall\, j \in \mathcal{J}, \tag{2.10}$$

$$x_i + p_i l_{xi} + q_i w_{xi} + r_i h_{xi} \le L_j + (1 - s_{ij})M, \qquad \forall\, i \in \mathcal{I}, j \in \mathcal{J}, \tag{2.11}$$

$$y_i + p_i l_{wi} + q_i w_{yi} + r_i h_{wi} \le W_j + (1 - s_{ij})M, \qquad \forall\, i \in \mathcal{I}, j \in \mathcal{J}, \tag{2.12}$$

$$z_i + p_i l_{zi} + q_i w_{zi} + r_i h_{zi} \leq H_j + (1 - s_{ij})M, \qquad \forall\, i \in \mathcal{I}, j \in \mathcal{J}, \tag{2.13}$$

$$l_{xi} + l_{yi} + l_{zi} = 1, \qquad \forall\, i \in \mathcal{I}, \tag{2.14}$$

$$w_{xi} + w_{yi} + w_{zi} = 1, \qquad \forall\, i \in \mathcal{I}, \tag{2.15}$$

$$h_{xi} + h_{yi} + h_{zi} = 1, \qquad \forall\, i \in \mathcal{I}, \tag{2.16}$$

$$l_{xi} + w_{xi} + h_{xi} = 1, \qquad \forall\, i \in \mathcal{I}, \tag{2.17}$$

$$l_{yi} + w_{yi} + h_{yi} = 1, \qquad \forall\, i \in \mathcal{I}, \tag{2.18}$$

$$l_{zi} + w_{zi} + h_{zi} = 1, \qquad \forall\, i \in \mathcal{I}, \tag{2.19}$$

$$x_i, y_i, z_i \geq 0, \qquad \forall\, i \in \mathcal{I}, \tag{2.20}$$

$$s_{ij} \in \{0, 1\}, \qquad \forall\, i \in \mathcal{I}, j \in \mathcal{J}, \tag{2.21}$$

$$n_j \in \{0, 1\}, \qquad \forall\, j \in \mathcal{J}, \tag{2.22}$$

$$l_{xi}, l_{yi}, l_{zi} \in \{0, 1\}, \qquad \forall\, i \in \mathcal{I}, \tag{2.23}$$

$$w_{xi}, w_{yi}, w_{zi} \in \{0, 1\}, \qquad \forall\, i \in \mathcal{I}, \tag{2.24}$$

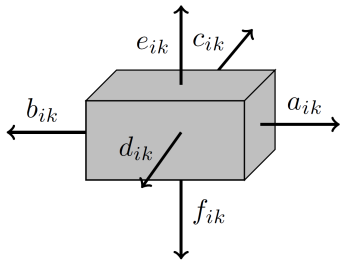
$$h_{xi}, h_{yi}, h_{zi} \in \{0, 1\}, \qquad \forall\, i \in \mathcal{I}, \tag{2.25}$$

$$a_{ik}, b_{ik}, c_{ik}, d_{ik}, e_{ik}, f_{ik} \in \{0, 1\}, \qquad \forall\, i, k \in \mathcal{I}, i < k \tag{2.26}$$

Objective function (2.1) minimizes the total empty space inside the used containers. Constraints (2.2)-(2.8) model the non-overlapping constraints by utilizing the relative position between boxes. Figure 2.1 illustrates all six possible regions (relative positions) to position a box adjacent to another without overlapping. For each pair of boxes i and k which are assigned to the same container, coordinates x_i, x_k, y_i, y_k, z_i and z_k are forced to satisfy the selected relative position of these boxes. For example, when a_{ik} is equal to one, then box i must be situated to the left of box k. Constraints (2.9) guarantee each box will be assigned to only one container, while Constraints (2.10) ensure containers containing at least one box are considered used. Constraints (2.11)-(2.13) guarantee that boxes are positioned completely inside their respective containers. Additionally, provided box rotations are permitted, each dimension of box i must be parallel to exactly one axis (X, Y or Z) of the container within which it is placed. These constraints are defined by Equations (2.14)-(2.19). Finally, Constraints (2.20)-(2.26) set the variables' domains.

Hifi et al. (2010) proposed a formulation for the 3D Input Minimization Problem without considering rotations. Its primary characteristic is that all containers are identical, thus the objective of the problem is to minimize the number of containers used. They do not specify whether the boxes considered are weakly

Figure 2.1: Relative positions for placing box k adjacent to box i.



or strongly heterogeneous and no real-world constraints are considered. The model is not solved in its entirety, being used in combination with a single container approach to find initial solutions for the problem. Then, by adding some valid inequalities also proposed in the paper, the formulation is applied to solve the subproblems of a tree search. It is important to mention that the valid inequalities proposed are only effective without considering rotations.

Tsai et al. (2015) proposed a linear formulation for the 3D Open Dimension Problem based on Chen et al. (1995)'s formulation. They consider a problem with a single container with no fixed dimensions whose objective is to minimize the container's total volume when all boxes are assigned to it. Regarding real-world constraints, only box rotations are considered. Although the main contribution of the paper is the linearization technique to generate the model, it also proposes a modification in the non-overlapping constraints to reduce the number of variables used.

Paquay et al. (2016) proposed a formulation for the Multiple Bin Size Bin Packing Problem (MBSBPP). They focus on air transportation, where the containers are truncated parallelepipeds. Many real-world constraints are introduced in the model such as orientation, stability, fragility of items, weight distribution and load bearing of the container. The main contribution of the paper is the proposal of a formulation for this very specific problem with the addition of all the constraints. In the model, the main difference comparing with the previous ones mentioned in this section is that instead of using only variables x_i, y_i, and z_i to represent a box position (front-left-bottom), variables corresponding to the rear-top-right corner (the corner opposite the box's origin) are also employed. Table 2.4 differentiates the primary characteristics of each formulation detailed throughout this section:

Table 2.4: Input minimization model characteristics.

	Container(s)				Additional
Formulation	Single	Varying Dimensions	Identical	Rotations	Constraints
Chen et al. (1995)		X		X	X
Hifi et al. (2010)			X		
Tsai et al. (2015)	X			X	
Paquay et al. (2016)		X		X	X

While the formulation proposed by Hifi et al. (2010) was employed on instances containing the highest number of boxes (up to 90), it is worth noting that the model itself was not solved in its entirety and the number of boxes per container was very low (at most six). For the remaining formulations, considering that the experiments executed were in their majority to validate them, Chen et al. (1995)'s formulation solved an instance with only 6 boxes, Tsai et al. (2015)'s solved instances with at most seven boxes while Paquay et al. (2016)'s was validated with instances considering at most 27 boxes, being able to solve instances with 18 boxes within the predefined time limit of one hour. The experiments were executed with different computer settings which hinders performance comparison.

Output Maximization mathematical models

Fasano (2004) proposed a formulation for the Output Maximization Problem without specifying the heterogeneity of the boxes neither the containers. The model's objective is to maximize the value of the boxes placed in a single container, where the value of a box equals its volume. They include orientation constraints directly into the formulation and also discuss how to consider balancing constraints. The main contribution of the paper is the proposal of a MILP-based heuristic to solve the problem. The general idea of the heuristic is to consider subsets of boxes and fix their relative positions, using the model to search for a feasible solution considering these positions. Aside from the objective function, the main difference between this model and Chen et al. (1995)'s is the representation of boxes by their center coordinates rather than by their corner (front-left-bottom).

Junqueira et al. (2012) proposed a different MILP formulation for an Output Maximization Problem. Originally not considering rotations, their formulation employs the concept of normal patterns. This technique reduces the set of possible coordinates a box can be placed by maintaining only coordinates that correspond to a multiple of a box dimension. By calculating all the possible box dimension's multiples within the container and combining them, instead of

having a continuous set of coordinates corresponding to the whole container, the new set is a reduced version that still guarantees the optimality of the solution. Another important difference is that the formulation proposed by Junqueira et al. (2012) assumes that boxes are grouped by type, meaning that those with identical dimensions ($p_i = p_k$, $q_i = q_k$ and $r_i = r_k$) are considered to be of the same type. This is useful to reduce the number of variables. Despite not considering rotations for the instances they addressed, the authors suggest that rotations may be incorporated by increasing the number of possible combinations based on the number of orientations permitted. For example, if two orientations are considered, each box would be included twice, one for each possible orientation and it would be enforced in the formulation that only one of the orientations can be placed. In further discussion Junqueira et al. (2012) also demonstrate how to include other real-world constraints such as stability of the cargo and container load bearing constraints.

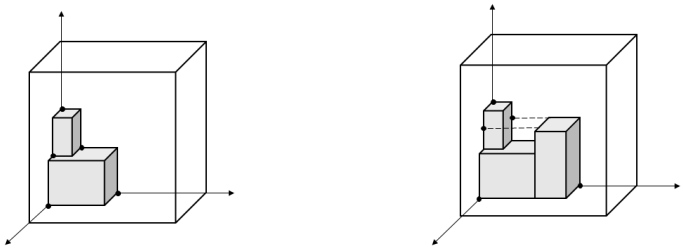
Concerning experiments, Fasano (2004)'s approach was successfully applied to instances with up to 20 boxes. Junqueira et al. (2012)'s formulation can solve instances with 68 boxes with no rotations allowed. It is important to mention that for these experiments the containers used were cubic ($L = W = H$) with small dimensions which contributed to reduce the number of discretised variables, resulting therefore in a smaller model. In other words, as mentioned in Junqueira et al. (2012), if the size of the container increases and the size of the boxes stay unaltered, the model would not be able to solve instances with the same number of boxes.

2.2.2 Combinatorial branch and bound

Martello, Pisinger, and Vigo (2000) proposed a combinatorial Branch and Bound algorithm for solving Output Maximization Problems considering one container (IIPP, SLOPP, SKP). Their method consists of ordering boxes by decreasing volume and generating a branching tree which combines all boxes in all feasible positions using a Depth-First Search (DFS) strategy. Feasible positions are generated by a procedure also proposed in the paper, known as corner points generation (Figure 2.2). Corner points are generated according to the position of the assigned boxes. Each placed box generates three corner points (above, to the right and in front) which correspond to the new possible placements for the following boxes. A dominance strategy is applied to the corner points generation to avoid overlapping. This strategy consists of checking if an existing corner point is affected by each new box to be placed. More specifically, for any corner point, if it cannot be guaranteed that any box can be positioned without overlapping, this corner point is either moved to a different coordinate if its

new position would be useful for a box placement or removed from the list of feasible points.

Figure 2.2: Corner points generation



(a) Placing a box. (b) Corner points update.

During the branching process, a pruning check is employed to evaluate if a new subtree should be investigated. First the solution's envelope volume is calculated as the volume of the assigned boxes summed together with empty spaces inside the container that can no longer be used to place new boxes. This envelope volume is then subtracted from the container volume, resulting in the container's *usable volume* (the volume where boxes can still be placed). This useful volume summed with the envelope volume corresponds to an upper bound concerning the maximum utilized volume within the container, which is subsequently compared against the current best solution found. If the upper bound is greater than the utilized volume in the current best solution, then the subtree should be explored (branched). Otherwise, the subtree is pruned.

Although this method was originally assumed to be complete, den Boef et al. (2005) demonstrated how only solutions with a specific placement setting, called robot packings, can have their optimal value found by Martello, Pisinger, and Vigo (2000)'s Branch and Bound with corner points. Robot packings can be generated by successively placing boxes beginning with the front-left-bottom corner of the container, with each new box being placed either in front of, to the right of, or above each of the previously-placed boxes. To be able to solve any placement setting, Martello, Pisinger, Vigo, et al. (2007) proposed a new version of their algorithm, aggregating the Branch and Bound together with a Constraint Programming approach that ensures the feasibility of a single bin's packing. It is important to highlight that robot packing instances are the most common in the literature, as mentioned by den Boef et al. (2005). Martello, Pisinger, Vigo, et al. (2007) confirmed this assumption by showing that only one out of the 764 instances solved to optimality had its best solution found by the Constraint Programming approach rather than by the Branch and Bound proposed by Martello, Pisinger, and Vigo (2000).

2.3 Problem adaptations

This section considers the following Output Maximization Problem variants: the SLOPP and the SKP. Since only those variants which consider a single container are being adapted, for the remainder of the chapter these problems will be referred to as Single Large Object Output Maximization (SLOOM) problems.

Concerning the inclusion of orientation constraints, for the formulation proposed by Hifi et al. (2010), which does not originally permit rotations, the same constraints as those proposed by Chen et al. (1995), namely Equations (2.14)-(2.19), were included in the model. The overlapping and bounding constraints were adapted following the same principle using Constraints (2.2)-(2.8) and (2.11)-(2.13) from Chen et al. (1995)'s formulation, respectively.

Junqueira et al. (2012), as mentioned in the previous section, suggested that for each allowed orientation a duplicate of the box is generated and the formulation guarantees that at most one of those duplicates will be placed in the container. The method proposed by Martello, Pisinger, and Vigo (2000) follows the same idea: each duplicate is added to the branching tree while making sure that exactly one box duplicate is considered in one branch.

2.3.1 Single Large Object Output Maximization adaptation

Concerning only the SLOOM, Fasano (2004), Junqueira et al. (2012) and Martello, Pisinger, and Vigo (2000) represent the approaches considered which were originally proposed for this variant. From these, Fasano (2004)'s formulation requires no modification, while the other two were adapted to utilize box duplicates as a means of accommodating rotations (see Section 2.3).

The formulations of Chen et al. (1995), Hifi et al. (2010), Tsai et al. (2015) and Paquay et al. (2016), originally proposed for Input Minimization Problems, require modifications concerning their objective functions and constraints. Input Minimization Problems consider multiple containers or a single container with at least one open dimension, with the objective function seeking to minimize the number of containers used, the waste of the used containers or the total volume of the container with open dimensions. In the SLOOM, however, the objective is to maximize the value of the placed boxes which in this chapter will be treated as the box volume. Take for example Objective Function (2.1), from Chen et al. (1995)'s formulation which was replaced by Objective Function (2.27):

$$\max \sum_{i \in I} p_i q_i r_i s_i \tag{2.27}$$

Here s_i is a variable which denotes whether box i is placed inside the container and $p_i q_i r_i$ corresponds to box i's volume. Therefore all objective functions from the Input Minimization Problems considered here were modified to Objective (2.27).

Concerning the problem constraints, the decision variable's index that indicates which container a box is assigned to was removed. For example, Constraints (2.8) from Chen et al. (1995) formulation are replaced by Constraints (2.28):

$$a_{ik} + b_{ik} + c_{ik} + d_{ik} + e_{ik} + f_{ik} \geq s_i + s_k - 1, \qquad \forall\, i, k \in \mathcal{I}, i < k \tag{2.28}$$

A similar principle is applied for bound Constraints (2.11)-(2.13). In this case not only variable s_i but also parameters L, W, H have one less index, as demonstrated by Equations (2.29)-(2.31).

$$x_i + p_i l_{xi} + q_i w_{xi} + r_i h_{xi} \leq L + (1 - s_i)M, \qquad \forall\, i \in \mathcal{I}, \tag{2.29}$$

$$y_i + p_i l_{yi} + q_i w_{yi} + r_i h_{yi} \leq W + (1 - s_i)M, \qquad \forall\, i \in \mathcal{I}, \tag{2.30}$$

$$z_i + p_i l_{zi} + q_i w_{zi} + r_i h_{zi} \leq H + (1 - s_i)M, \qquad \forall\, i \in \mathcal{I} \tag{2.31}$$

The same modifications were applied to the formulations proposed by Hifi et al. (2010), Paquay et al. (2016) and Tsai et al. (2015). The adapted approaches are henceforth referred to using the abbreviations provided by Table 2.5 so as to facilitate a more natural reading of the current chapter.

2.4 Computational results

Experiments were performed considering the SLOOM. Instances were taken from the literature of 3D packing problems in addition to others created using the generator proposed by E. Silva, Oliveira, et al. (2014).

Table 2.5: SLOOM adaptations.

Literature	Adaptation
Chen et al. (1995)	OM-Chen
Martello, Pisinger, and Vigo (2000)	OM-Martello
Fasano (2004)	OM-Fasano
Hifi et al. (2010)	OM-Hifi
Junqueira et al. (2012)	OM-Junqueira
Tsai et al. (2015)	OM-Tsai
Paquay et al. (2016)	OM-Paquay

Mathematical formulations were evaluated for all instance sets on both CPLEX 12.6.3 and Gurobi 6.5.2 using their default settings. In terms of implementation, all formulations and OM-Martello were fully implemented with the programming language Java 8 and the Java Programming Language Compiler (javac) from its original development kit (JDK). Due to the extremely large number of experiments performed (approximately 15600h of CPU time), only the best solutions, together with the corresponding solver, are displayed.

2.4.1 Instances

Instances from the literature were: LN taken from Loh and Nee (1992), BR taken from Bischoff and Ratcliff (1995), SA taken from Allen et al. (2012) and CHEN, TSAI and TSAIRW, taken from Tsai et al. (2015).

The CHEN, TSAI and TSAIRW instances are all considered small in size given they only contain 4-12 boxes. These small instances permit all six possible orientations.

The SA instance set does not permit rotations, has 20 sets of instances, each of which contains 20 instances sharing the same characteristic: the first instance has five boxes, the second has ten boxes and so on until the last instance which has 100 boxes. Furthermore, it is known that the optimal solutions for all these instances have all their associated boxes placed inside the container with exactly 100% volume occupation.

The LN instance set, whose instances contain 100 to 250 boxes, considers the height of each box as fixed, thereby only permitting rotations regarding the width and length of boxes. Finally, the BR instance set consists of the first seven instance classes from Bischoff and Ratcliff (1995), BR1–BR7, with each class containing 100 instances. Each BR instance has between 69 to 476 with either two, four or all six orientations permitted.

After analyzing the results obtained considering instances from the literature, it was observed that the instances are either too small being easily solved or too large in terms of number of boxes, resulting in poor quality solutions. Moreover, the special case of the SA instance set in which containers have 100% occupation is impractical in real-world situations. These characteristics complicate the performance and behavior analysis of exact methods, motivating the need for medium-size instances.

Therefore, new instance sets were created using the 2DCPackGen, introduced by E. Silva, Oliveira, et al. (2014), a 2D and 3D rectangular Cutting and Packing instance generator, according to the typology proposed by Wäscher et al. (2007). The 2DCPackGen utilize various parameters for the generation of boxes and containers: their dimensions, configurations and number of boxes. Challenging instances were created with a scaling behavior for further investigating the performance of the formulations and methods considered.

Three instance classes were created: ScaleSetA (SSA), ScaleSetB (SSB) and ScaleSetC (SSC). SSA consists of four sets of 30 instances each, with the first instance of each set containing just a single box and each subsequent instance adding an additional different one until finally the last instance of each set has 30 different boxes (for these instances this also means 30 different box types). A box being classified as different from another implies that at least one of its dimensions differs from all other boxes. Even though solving an instance with one box is trivial, we maintained it during the experiments for the sole purpose of providing a full curve of the scaling behavior for the solution of these sets (which will be provided in the next section). The parameters employed to generate SSA are that boxes may be either small and square-shaped or long and narrow, while containers are limited to a small and square configuration. All six possible orientations are allowed, thereby guaranteeing that all box positions are considered.

Each of these 30 instances was generated four times resulting in the following classes: SSA60, SSA70, SSA80 and SSA90. The only difference in these sets is the container dimension, such that the number associated with each class represents the occupation percentage of the boxes as a proportion of its respective container's volume. For example, SSA60 corresponds to instances where the volume of the corresponding boxes equals approximately 60% of the container's volume, while each instance from set SSA70 contains boxes which the sum of their volume is equivalent to approximately 70% of the container occupation. In other words, the difference between sets SSA60, SSA70, SSA80 and SSA90 are the container dimensions, while the boxes used are the same. The idea behind generating these instance sets is to evaluate the behavior of the methods when the tightness of the container varies. In the experiments executed, a tighter container corresponds to a container which has smaller dimensions compared to

a container from a previous test considering the same set of boxes to be placed. For example, for class SSA, SSA70 has a container tighter than SSA60.

It is important to recall that, given how the SLOOM seeks to maximize the value of the items assigned to a single container, one could assume that for instances with 60% - 90% container occupation this entails that all boxes are capable of fitting inside the container. When considering only the volume of the boxes, the problem would be considered as one-dimensional. In this case one would consider that the boxes are "liquid", instead of solid which gives the aforementioned conclusion. However, in reality, the three solid dimensions of the boxes make this assumption not true for most cases. For example, a box could have a length bigger than all three dimensions of the container and at the same time have such small width and height that its volume is smaller than the container's. The same holds for the BR instances, the volume of whose constituent boxes is also less than the volume of the containers, however the best approaches in the literature have thus far been unable to find patterns that fit all boxes inside the container.

SSB and SSC have similar characteristics to SSA concerning number of instances, boxes per instances and average percentage of container volume occupation. The only significant difference concerns the parameters used to generate these instance sets. For SSB instances, boxes configuration may be either short and tall or long and tall, while the containers are always small and square-shaped. Meanwhile, for the SSC instance sets, boxes are characterized as being either short and of average width or long and of average width, with the containers again being small and square-shaped.

Details concerning the number of instances per set/total, minimum and maximum number of boxes, box types, number of orientations permitted and type of problem (SLOPP or SKP) are provided in Table 2.6:

Table 2.6: Characteristics of the considered instance sets.

Set	Instances per set /total	Boxes	Types	Rot	Problem	Literature
CHEN	4	4-7	4-7	6	SKP	Tsai et al. (2015)
TSAI	4	8-12	3-5	6	SKP	Tsai et al. (2015)
TSAIRW	4	4-7	4-6	6	SKP	Tsai et al. (2015)
SA	20/400	5-100	5-100	0	SKP	Allen et al. (2012)
LN	15	100-250	6-10	2	SLOPP	Loh and Nee (1992)
BR	100/700	69-476	3-20	2-6	SLOPP	Bischoff and Ratcliff (1995)
SSA	30/120	1-30	1-30	6	SKP	Generated
SSB	30/120	1-30	1-30	6	SKP	Generated
SSC	30/120	1-30	1-30	6	SKP	Generated

2.4.2 Single Large Object Output Maximization

Experiments were executed for each instance with a runtime limit of 3600s on an Intel(R) Xeon(R) E5-2640 CPU 2.6GHz with 64GB of RAM. Preliminary experiments which compared solutions acquired using 4 and 16 threads signaled no significant difference concerning solution quality. Given this knowledge and the extremely large number of experiments (over 15,000h of experiments if executed sequentially), solvers were consequently executed using 4 threads.

Table 2.7 details the performance of the compared methods on the small instance sets CHEN, TSAI and TSAIRW. The table shows the solver which obtained the best results for the mathematical models, the number of optimal solutions found and the average time required to find these solutions.

Table 2.7: CHEN, TSAI and TSAIRW instances.

Adaptation	Best Solver	CHEN		TSAI		TSAIRW	
		Opt sols	Avg t(s)	Opt sols	Avg t(s)	Opt sols	Avg t(s)
OM-Paquay	Gurobi 6.5.2	**4**	0.02	**4**	0.15	**4**	0.08
OM-Hifi	Gurobi 6.5.2	**4**	0.02	**4**	0.18	**4**	0.28
OM-Chen	Gurobi 6.5.2	**4**	0.04	**4**	0.29	**4**	0.33
OM-Fasano	Gurobi 6.5.2	**4**	0.04	**4**	0.26	**4**	0.43
OM-Tsai	Gurobi 6.5.2	**4**	0.10	**4**	1.05	**4**	0.21
OM-Martello	-	**4**	0.03	**4**	1.29	**4**	0.14
OM-Junqueira	Gurobi 6.5.2	**4**	27.30	**4**	0.05	1	280.59

It can be seen that for the CHEN and TSAI instance sets all solutions were computed within short runtimes. However, for the TSAIRW instances, OM-Junqueira required a considerably longer runtime to solve the smallest instance, did not find an optimal solution for the second instance within the runtime limit and was unable to generate any feasible solution for the last two instances due to insufficient memory when generating the discretized model. By contrast, all other methods solved all the TSAIRW instances in less than a second on average.

Tables 2.8 and 2.9 present results for the SA instances, providing each method's performance concerning optimal solutions and feasible solutions not proven optimal within the runtime limit, respectively. Considering that the objective function of these problems is to maximize the volume occupation of the container, Table 2.9 provides the average percentage of container occupation for the non-optimal solutions. The first remark worth making is that all methods generated feasible solutions for all instances. Another important remark is that, to limit the size and number of tables in this chapter, only results for the first three

sets of instances are provided since results for the remaining sets are similar to those of the third instance set (SA15).

Table 2.8: SA instances (optimal solutions).

Adaptation	Best Solver	Optimal solutions (20)			Average time (s)		
		SA5	SA10	SA15	SA5	SA10	SA15
OM-Junqueira	CPLEX 12.6.3	**20**	**20**	**6**	0.1	52.9	353.3
OM-Chen	Gurobi 6.5.2	**20**	5	5	462.3	73.1	250.3
OM-Hifi	Gurobi 6.5.2	**20**	4	3	692.1	810.7	0.6
OM-Fasano	Gurobi 6.5.2	19	5	5	594.0	40.5	104.5
OM-Tsai	Gurobi 6.5.2	9	3	3	616.0	0.9	2.9
OM-Paquay	Gurobi 6.5.2	8	5	5	203.3	681.1	703.5
OM-Martello	-	2	2	2	0.2	0.1	0.1

Table 2.9: SA instances (non-optimal solutions).

Adaptation	Best Solver	Non-optimal sol (20)			Avg % container occ		
		SA5	SA10	SA15	SA5	SA10	SA15
OM-Junqueira	CPLEX 12.6.3	**-**	**-**	**14**	**-**	**-**	80.1
OM-Chen	Gurobi 6.5.2	**-**	15	15	**-**	98.1	94.0
OM-Hifi	Gurobi 6.5.2	**-**	16	17	**-**	92.2	89.7
OM-Fasano	Gurobi 6.5.2	1	15	15	95.2	93.7	92.6
OM-Tsai	Gurobi 6.5.2	11	17	17	95.6	87.8	84.1
OM-Paquay	Gurobi 6.5.2	12	15	15	96.7	92.1	88.7
OM-Martello	-	18	18	18	86.1	73.2	69.7

OM-Junqueira outperformed all other methods for all three SA instance sets. Although the boxes considered are strongly heterogeneous (SKP), the discretized approach performs very well provided that the boxes dimensions consist of multiples of one another in most cases. The more homogeneous the dimensions are, the more similar the normal patterns will be, thereby preventing the model from escalating in size during MILP generation and in turn avoiding memory problems. For this type of problem such discretization is useful for finding good solutions in a short amount of time compared against the other methods.

Concerning the solvers, OM-Junqueira was the only one for which CPLEX outperformed Gurobi. Both obtained solutions with similar objective values, however CPLEX found them in less time. Meanwhile the other adaptations did not generate solutions with the same quality, the primary reason being how tight the optimal solutions are (100% of volume occupation, as detailed in Section 2.4.1), thereby making it difficult to solve for instances of this type with a considerable number of boxes.

Table 2.10 presents the results for the LN instances. Each method's performance concerning optimal solutions is detailed together with the results concerning solutions not proven optimal within the runtime limit. Column $Nsols$

corresponds to the number of solutions found, proved optimal or just feasible and *Avg occ* corresponds to the average percentage of container occupation.

Table 2.10: LN instances.

Adaptation	Best Solver	Optimal solutions		Non-optimal solutions	
		Nsols	Avg t(s)	Nsols	Avg occ (%)
OM-Chen	Gurobi 6.5.2	**6**	274.6	9	67.6
OM-Martello	-	**5**	2.8	10	55.8
OM-Tsai	Gurobi 6.5.2	4	568.1	11	63.7
OM-Fasano	Gurobi 6.5.2	4	1433.3	11	41.7
OM-Hifi	Gurobi 6.5.2	4	1502.7	11	40.1
OM-Paquay	Gurobi 6.5.2	1	92.6	14	42.1
OM-Junqueira	-	0	-	0	-

The low number of optimal solutions indicate that the LN instances, having up to 250 boxes, are challenging to solve. OM-Chen obtained the best results, solving 6 out of 15 instances to optimality. OM-Junqueira was unable to find any solution for the problem due to insufficient memory for all 15 instances, given the number of boxes and (consequently) the size of the model's required discretization. OM-Martello was able to generate five optimal solutions in a short amount of time, given that its depth-first search (DFS) is generally capable of placing all the boxes quickly in instances whose optimal solutions are not tight in terms of container occupation (under 70%).

The BR instances, the largest ones considered (which contain 69 - 476 boxes), were so challenging in terms of size that no method was capable of solving any of them to optimality. Therefore, Table 2.11 presents the average percentage of container occupation for each solution found.

Table 2.11: BR instances.

Adaptation	Best Solver	Average % container occupation						
		BR1	BR2	BR3	BR4	BR5	BR6	BR7
OM-Tsai	Gurobi 6.5.2	**73.0**	**72.4**	**72.1**	**71.9**	**70.9**	**70.5**	**70.3**
OM-Chen	Gurobi 6.5.2	67.7	67.3	65.7	66.3	65.7	65.4	63.2
OM-Martello	-	63.4	59.1	56.7	56.4	56.2	54.5	53.3
OM-Hifi	Gurobi 6.5.2	56.4	56.8	55.7	56.2	56.5	55.6	55.3
OM-Paquay	Gurobi 6.5.2	49.5	51.8	53.7	52.5	54.5	55.7	55.6
OM-Fasano	Gurobi 6.5.2	43.2	45.1	44.9	45.3	43.8	45.4	45.8
OM-Junqueira	-	-	-	-	-	-	-	-

For the BR instances, results demonstrate how OM-Tsai outperforms the other adaptations, having the highest average percentage of container occupation for all seven instance sets. OM-Junqueira had memory issues for all instances due to the number of boxes considered resulting in a prohibitively large discretization of the model.

2.4.3 Generated Instances

Each test was restricted to 5h (18000s) of runtime for each ScaleSet instance, with each solver executed using 4 threads on an Intel(R) Core(TM) i7-2600 CPU 3.40GHz with 8GB of RAM. Graphs are employed rather than tables when presenting results in order to intuitively visualize scaling performance.

Figure 2.3: SSA60 and SSA70 instances.

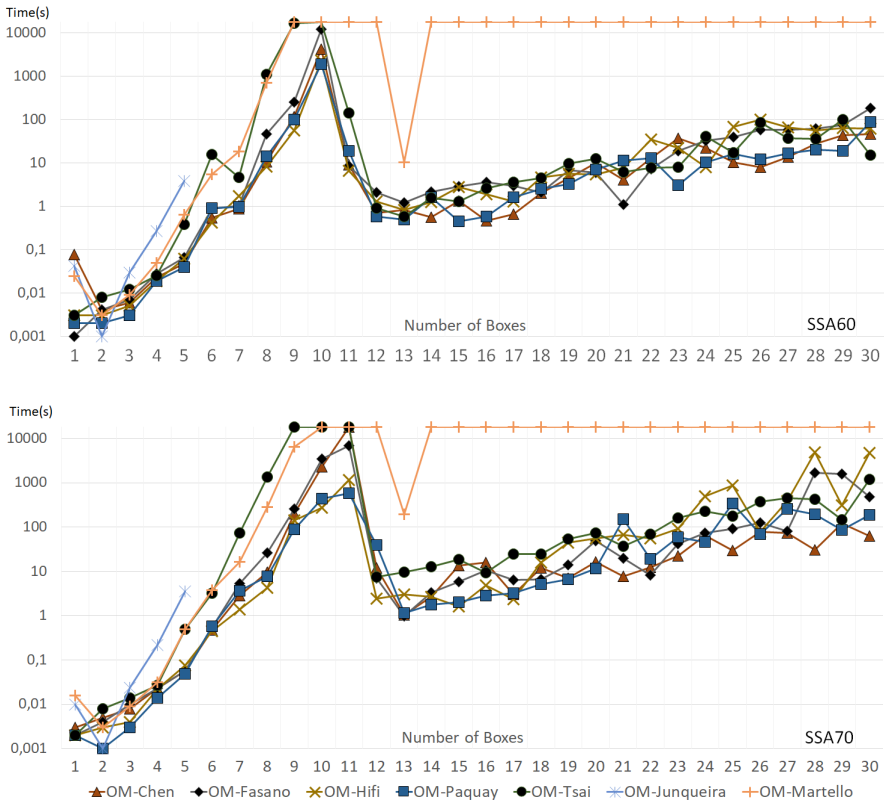
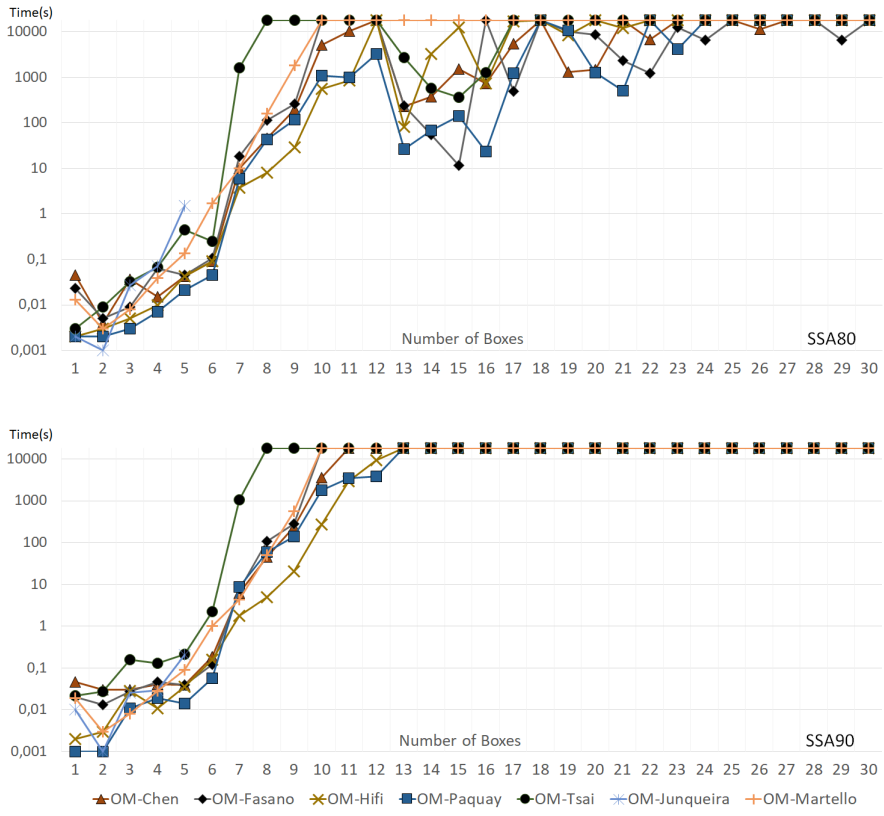


Figure 2.3 illustrates the execution time for each method when solving the SSA60 and SSA70 instance sets, respectively. Note that the vertical axis presents the execution time in logarithmic scale while the horizontal axis corresponds to the instances (1 - 30), with the number of boxes being associated with each instance displayed. For example, instance SSA60_11 is the eleventh instance and contains 11 boxes (also 11 box types). One can clearly see the increase in execution time as the number of boxes increases.

One can also see how the graph naturally divides itself into two distinct regions. The first region, spanning instances 2 - 10, is characterized by having optimal solutions in which not all the boxes can fit inside the container. The dramatic increase in runtime shows how challenging it is to prove the optimality of solutions where boxes are left outside the container. The second region (instances 13 - 30) corresponds to instances in which optimal solutions have all boxes placed inside the container. The less dramatic increase in runtime demonstrates how it is easier to prove optimality for instances in which all boxes can be placed inside the container. This is expected given that solutions cannot be further improved once all boxes are successfully placed inside the container. Although the second region does not belong to the scope of the problems considered, by finding solutions that allocate **all** boxes inside, it is remarkable to show the difference in difficulty for the model to find these solutions.

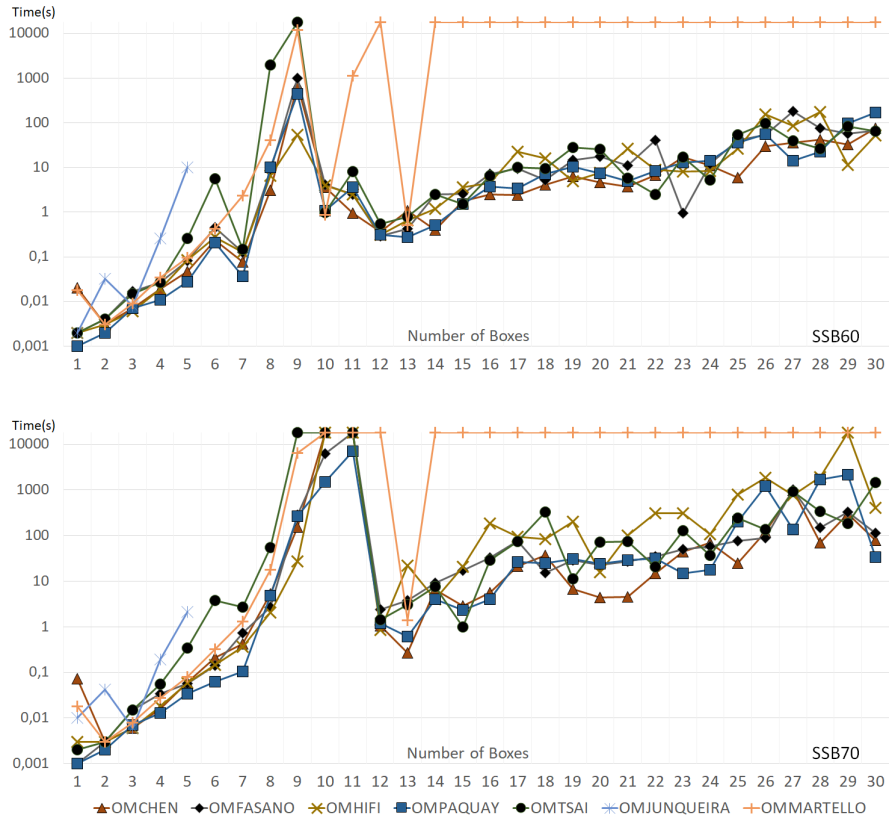
Figure 2.4: SSA80 and SSA90 instances.



The graph of instance set SSA70 shows the influence of employing tighter containers compared to the previous instance set. It can be observed how instances 2 - 11 (instead of 2 - 10 from SSA60) have optimal solutions in which not all the boxes can be placed inside the container. For the remaining instances (12 - 30), although easier to solve, the execution time is also affected by increased container tightness.

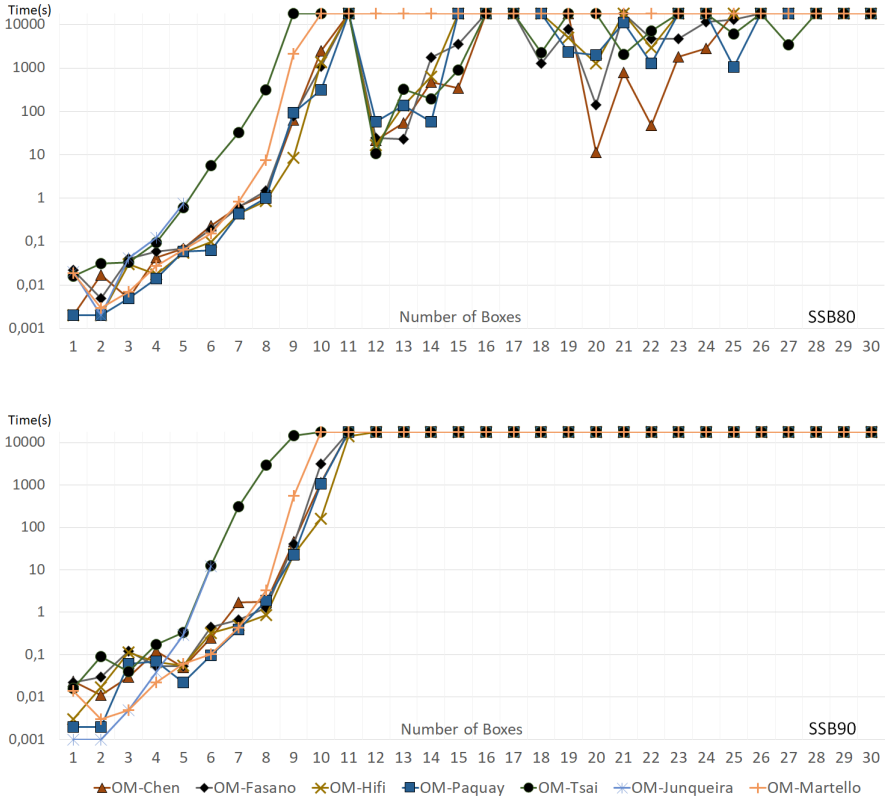
Figure 2.4 shows the solutions for a container 10% tighter than the previous instance set. For the SSA80, instances 2 to 12 have optimal solutions which do not place all the boxes inside the container. It can be seen that for containers with an average occupation of 80%, the behavior of the considered methods becomes very similar to their behavior with respect to the largest instances from the literature (LN and BR) that have no scaling characteristic.

Figure 2.5: SSB60 and SSB70 instances.



Finally, the SSA90 solution graph provides the results for the tightest SSA instances. From this set, the largest instance with a proven optimal solution has 12 boxes. For the remaining instances, it was not possible to determine if all boxes can be placed inside the container. This solution is only proven optimal by OM-Hifi and OM-Paquay while all other approaches reach the runtime limit. This experiment highlights the limitations of the adapted formulations, which are unable to prove optimality for instances with more than 12 boxes when considering tight containers.

Figure 2.6: SSB80 and SSB90 instances.

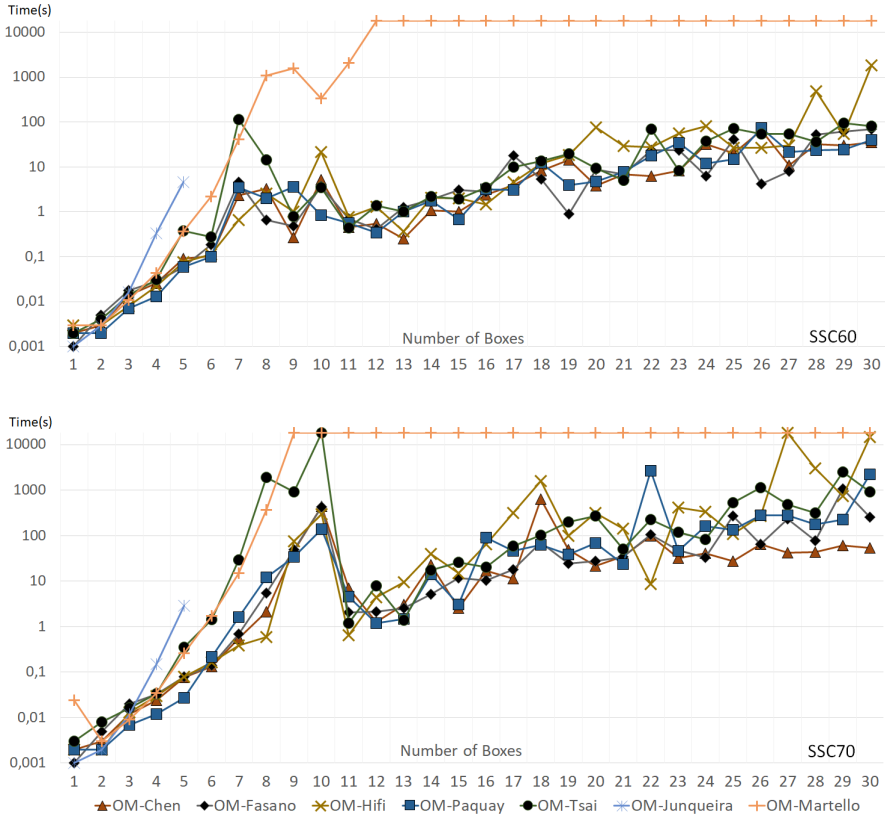


A different box setting is considered for the SSB instance set: boxes are either short and tall or long and tall. Figure 2.5 illustrates the performance of the methods for instance sets SSB60 and SSB70, respectively, accross the same range of container occupation rates as before.

Solution values for the SSB60 exhibit a similar behavior as the SSA60 with two distinct regions: from 2 to 9 boxes and from 10 to 30. The first region has a very steep increase in time and corresponds to instances for which optimal solutions have a number of boxes not placed inside the container while the second region corresponds to instances whose optimal solutions have all boxes placed inside the container. Comparing both scaling instance sets, SSA60 and SSB60, with respect to the average execution time of the proposed methods, these graphs reveal that the settings associated with instance set SSB provide better packing patterns and result in shorter execution times.

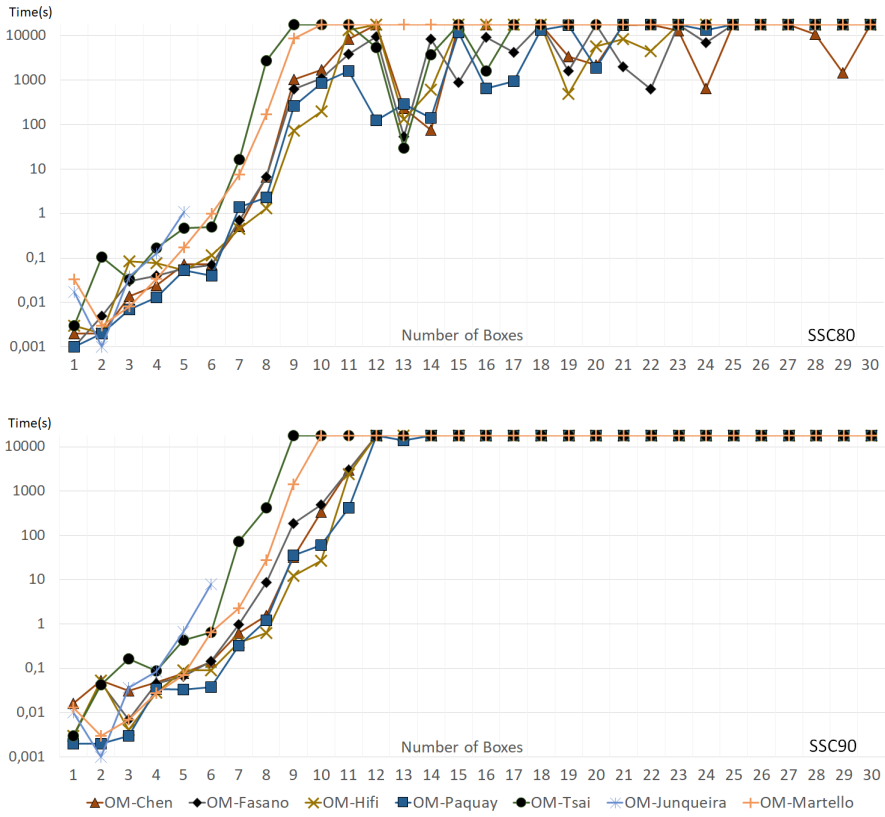
Concerning set SSB70, container tightness influences the solution in a manner similar to set SSA. As the container gets tighter, execution time increases and the two regions in the graph become more distinct (Figure 2.6).

Figure 2.7: SSC60 and SSC70 instances.



The same behavior is observed for the SSC instances (Figures 2.7 and 2.8). The different shape of boxes resulted in better solutions, particularly for instance sets SSC60 and SSC70, (Figure 2.7) which have the shortest average runtimes in comparison to instances from the SSA and SSB sets. Meanwhile, for instance sets SSC80 and SSC90 (Figure 2.8) container tightness impacts the instances enough to result in significantly longer execution times when compared against the SSC60 and SSC70 sets.

Figure 2.8: SSC80 and SSC90 instances.



Three primary conclusions are evident with regard to the scaling of different methods. First, OM-Chen and OM-Paquay outperform the other methods, on average, finding better or similar solutions within a shorter execution time for all the analyzed instances. Second, OM-Junqueira solves only small instances (up to five boxes), presenting memory issues for the remaining instances. This behavior is a consequence of considering all six possible orientations, which in

the discretized model results in five duplicates of the original boxes, significantly increasing the size of the proposed model and, in turn, the amount of memory required.

Finally, OM-Martello has a similar behavior across all instance sets: (i) an increase in runtime occurs when obtaining optimal solutions with boxes outside the container, (ii) the runtime limit is reached for the majority of the remaining instances and (iii) runtime is also affected by the number of permitted box orientations, since duplicating boxes ultimately enlarges the search tree.

2.4.4 Rotations

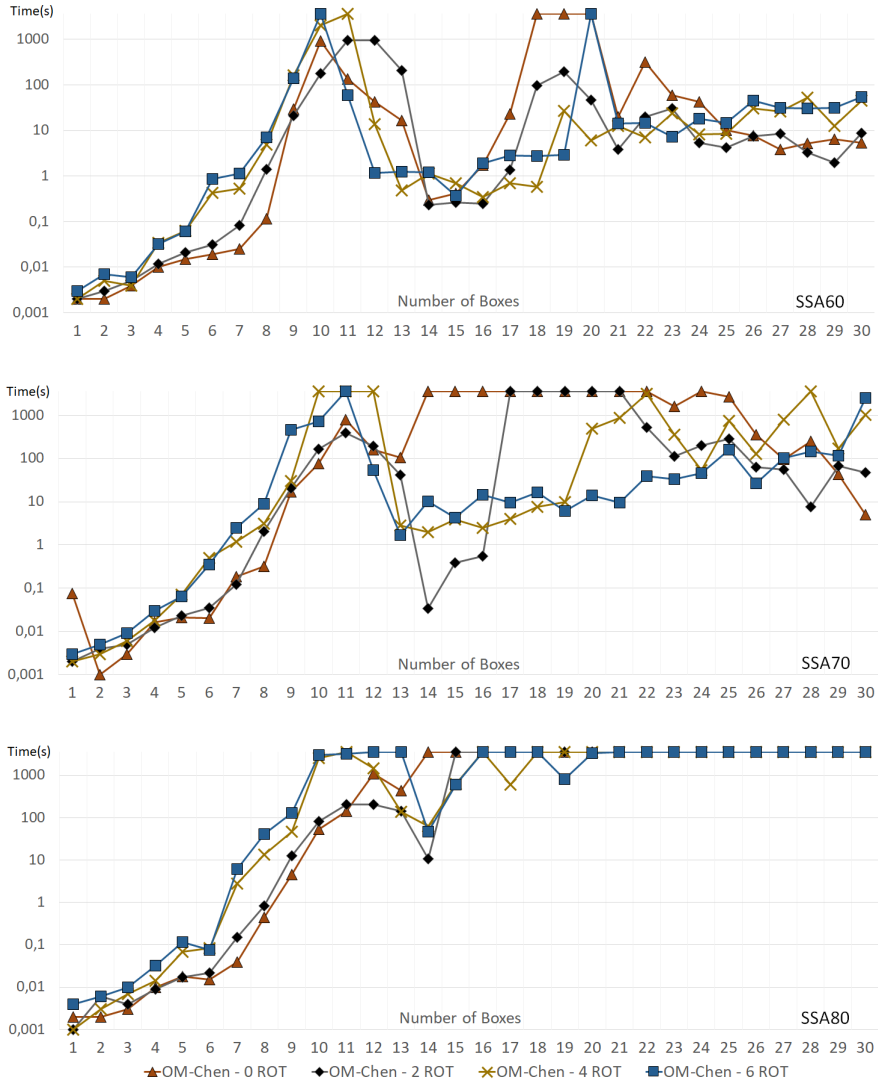
Experiments were performed using the SSA instances in order to assess the influence of rotations upon the general performance of the considered methods. These instances were modified so that each box was allowed zero, two, four or six orientations, generating four new instance sets. OM-Chen was employed using the Gurobi solver and with a runtime limit of 1h. The SSA60, SSA70 and SSA80 instances were tested, respectively, with the results illustrated by Figure 2.9.

The first insight provided by these figures is that there is no clear number of orientations which outperforms the others on the instance sets being tested. The number of orientations which, on average, provided the best results in terms of number of boxes placed and container volume occupied (%) within the runtime limit was 2 for SSA60, 6 for SSA70 and again 2 for SSA80. This behavior demonstrates how the number of permitted orientations should also be considered as an important component that influences the solution quality.

The number of allowed orientations corresponds to the possibilities of different displays (packing patterns) of the boxes. In other words, the larger the number of orientations, the larger the number of possible packing patterns. The most obvious advantage of generating more packing patterns is the chance of placing more boxes inside the container. The problem is that increasing the number of possible orientations increases the number of variables, which affects the solver performance.

One must also consider how many orientations are really necessary to find the best solutions given that all 6 orientations are not always required for placing the highest number of boxes. Considering those instances from SSA60 containing 21 to 30 boxes, for example, solutions which placed all the boxes were found even when no rotations were permitted. The same behavior occurred for instances with 25 to 30 boxes in the SSA70 instance set. In these situations

Figure 2.9: Orientation comparison for the SSA60, SSA70 and SSA80 instances.



the extra variables required to consider multiple orientations are not necessary, yet including them generally results in significant increase in execution time.

2.4.5 Discussion

After analyzing all executions, various interesting conclusions emerge. Regarding the linear relaxation of the formulations, OM-Junqueira, the discretized formulation, provides the best upper bounds from all adaptations. All others have similar linear relaxation value: the sum of the boxes volume (value), which is only useful for instances whose optimal solutions place all boxes inside the container.

Despite OM-Junqueira having a better linear relaxation, the normal patterns approach is effective for a very limited set of instances. This can be seen even with the smallest instances (TSAIRW). This formulation was able to solve only one instance with four boxes within the time limit, while instances with six and seven boxes presented insufficient memory to solve them. The reason for this is that the more heterogeneous the boxes, the larger the normal patterns set for each axis becomes, increasing considerably the size of the model to be solved.

Regarding the remaining formulations, the binary variables that correspond to the relative position of the boxes explain the poor linear relaxations. Different experiments were performed with the formulations, manipulating the value of subsets of all variables. In all cases where the value of the relative positions' variables were not fixed the linear relaxation always corresponded to the sum of the volumes of all boxes. The reason behind this is the manner in which the constraints regarding relative positions are created. Whenever relaxed, these binary variables can always generate a combination of non-integer values which will allow the model to be feasible with all the boxes "placed inside" the container. Unfortunately we were unable to determine an alternative way in which we could formulate these constraints in a way that would restrict at least a subset of these variables to always assume a binary value. However, these experiments also indicated that for each pair of boxes placed inside the container when at least one of the relative positions' variables is active the solver is able to generate a feasible solution, if existing, in a matter of seconds even for large instances such as LN.

The difficulty of solving the problem (as shown in Section 2.4) together with the poor relaxation of all formulations, except OM-Junqueira, leaves their solution process at the mercy of the randomness concerning which heuristic methods are applied within the solvers. Solutions for instances LN and BR, for example, were result of these heuristic procedures applied by the solvers given the high number of boxes in these instances. This information is confirmed by analyzing the output files provided by the solvers. A good example of the degree to which

randomness affects the solution is that changing the random seed[1] within the solvers influenced execution time significantly.

Another analysis concerns the size of the formulations. Table 2.12 provides the number of variables and constraints for each adaptation, where n corresponds to the number of boxes. For OM-Junqueira, given that the boxes are grouped by type according to their dimensions (Section 2.2.1), t corresponds to the number of box types and x, y and z correspond to the number of normal patterns in each axis of the container.

Table 2.12: Size of the adapted formulations.

Adaptation	Variables	Constraints
OM-Chen	$3n^2 + 10n$	$(13n^2 + 31n)/2$
OM-Fasano	$3n^2 + 10n$	$(13n^2 + 37n)/2$
OM-Hifi	$3n^2 + 10n$	$(13n^2 + 31n)/2$
OM-Junqueira	$12txyz$	$12txyz + xyz + 6t$
OM-Tsai	$3n^2 + 13n$	$(17n^2 + 29n)/2$
OM-Paquay	$3n^2 + 13n$	$(17n^2 + 39n)/2$

Comparing the size of the models, OM-Chen and OM-Hifi have the smallest number of variables and constraints. The fact that OM-Chen has the smallest model could explain their best overall performance on small, medium and large instances. Regarding the OM-Tsai and OM-Paquay adaptations which also performed well, on average, the reasoning can be explained by the different way their constraints are created, with OM-Paquay considering both corners of the boxes (front-left-bottom and rear-top-right) and OM-Tsai modifying the relative position constraints to use a smaller set of binary variables. It is important to mention that even though both adaptations performed well for small and medium instances, OM-Paquay's largest size affects its performance for the largest instances (BR) where OM-Tsai outperformed OM-Paquay.

A noticeable similarity between all formulations is that, with the exception of Paquay et al. (2016) when solving the MBSBPP and Tsai et al. (2015) when solving the ODP, none of the models were created taking into account a specific 3D Packing Problem variant shown in Table 2.1. This generalization of the problems to be solved does not allow the models to take into account problem-specific features. For example, the model by Junqueira et al. (2012), which considers boxes by type, is more effective for instances containing boxes with similar dimensions, which could be explored to solve only the SLOPP or IIPP (not considered in this research). The IIPP (Identical Item Packing Problem) considers only sets of identical boxes to be placed in a single container. Another approach that could be considered for weakly heterogeneous boxes would be to

[1] A random value chosen by the solver which any changes corresponds to perturbations in the solution paths.

solve the problem by replicating packing patterns to fill the container the same way as it is already done in the literature concerning 1D and 2D Cutting and Packing problems.

Experiments concerning OM-Martello also provided interesting conclusions. The Branch and Bound obtained optimal solutions for small and medium instances (some LN instances, for example) despite being heavily affected by rotations and tight containers. During implementation it was observed that improvements are possible by incrementally calculating the corner points, instead of recalculating each corner point in each node of the branching tree, as well as by grouping together boxes by type (boxes with identical dimensions), using the same strategy as OM-Junqueira.

A final comparison concerning the influence of rotations in some of the SSA instances revealed the importance of choosing the adequate number of permitted orientations. The main problem related to rotations is that no previous knowledge concerning optimal solutions is known, making it difficult to predefine the number of orientations such that it is not needlessly high, thereby introducing more variables than are necessary.

2.5 Conclusions

This chapter adapted the most significant exact methods proposed in the literature for 3D packing problems to solve Output Maximization Problems considering a single container. In total, six mathematical formulations and an additional specialized Branch and Bound method were re-implemented and the results from 15600h of experiments on up-to-date hardware were compared.

Instances from the literature were unsuitable for distinguishing the performance of the different approaches. These instances exhibit extreme characteristics by either considering too few or too many boxes. Therefore, experiments were conducted considering new instances produced using a generator from the literature, enabling us to understand the scaling behavior and performance limits of the adapted approaches.

Many conclusions were reached, each of which provides interesting research possibilities for exact 3D packing problems. The primary conclusion is that it still remains very difficult to optimally solve even medium-sized instances. The bottleneck is proving the optimality of solutions. Another impacting factor is the tightness of the container. The tighter the container, the harder it is to find solutions within acceptable runtimes. Additionally, the number of boxes

affects the performance of the solution method: increasing the number of boxes generally increases the runtime, as expected.

It was also revealed how the linear relaxation of all the non-discretized formulations are ineffective and depends entirely on the binary variables that correspond to the relative positions of each pair of boxes. Not only the relaxation, but the overall performance of the formulations is completely dependent on how efficient a solution method is at finding the best combination of these positions for the boxes. This indicates an extremely worthwhile avenue for future research: rather than approaching this problem focusing on a direct representation of the coordinates of the boxes, consider an indirect approach of a solution space of possible relative positions between the boxes. This strategy is further investigated in the next chapter, where it is used to efficiently solve a realistic variant of the problem.

Furthermore, important discussions concerning the number of orientations allowed and the initial upper bound for these problems were conducted, providing further insight regarding research on these topics. Future research including investigation of extra information during the preprocessing could prove very useful in terms of determining the necessary number of orientations instead of always considering the maximum number permitted by an instance.

Other research to be explored concerns the design of new formulations, given that currently only two approaches exist: the discretized one proposed by Junqueira et al. (2012) and the continous one originally proposed by Chen et al. (1995), which has since been further modified in the literature. Formulations focused on solving specific variants of the problem such as the SLOPP and which make use of the characteristics of the boxes in this type of problem will likely produce methods that outperform the existing ones.

Additional future research should include similar analyses considering different variants of 3D packing problems such as Input Minimization variants. The approaches investigated during the present chapter can be adapted to these variants and compared against one another in order to verify whether the same conclusions hold. Another topic which should be further investigated is the calculation of tighter initial upper bounds for these problem variants (SLOPP and SKP). All the aforementioned topics provide research directions which may result in significant performance improvements for old and new solution approaches alike.

Chapter 3

A matheuristic framework for the three-dimensional Single Large Object Placement Problem with practical constraints

After thoroughly reviewing the literature on heuristic methods and conducting our own review of exact methods for solving regular 3D packing problems, we are now ready to address realistic problems in an efficient manner. In other words: providing high quality solutions under a wide range of scenarios. Considering existing problem variants, we begin by addressing the Single Large Object Placement Problem. Although this is one of the most studied packing problems in academia, very few publications have considered more than the three basic constraints: non-overlap, orientation and stability. In order to make the problem more realistic, various additional constraints can be considered depending on the specific real-world application being solved, such as load balance, stacking and positioning.

When analyzing heuristic methods, we realized that only part of our goal could be achieved: good quality solutions in very specific scenarios. On the other hand, when considering exact methods, although it was easier to create a flexible method capable of adapting to different combinations of constraints,

the solutions ultimately obtained are of low quality. This encouraged us to develop a compromise by bringing together the advantages of each approach and developing a matheuristic approach.

This matheuristic is capable of handling combinations of the following real-world constraints: orientation, load balance, loading priorities, positioning, stability, stacking and weight limit. The results demonstrate the robustness of the proposed matheuristic when solving different combinations of constraints. It is also shown how to extend this method to consider variants with multiple containers in order to show how simple it is to adapt the proposed solution approach to other problems.

3.1 Introduction

Container Loading Problems (CLP) are a set of well-known combinatorial optimization problems that are of crucial relevance in the logistics industry. Their significance is threefold in nature: company profitability, employee safety and environmental impact. Considering all the categories proposed by Wäscher et al. (2007), this chapter focuses on solving the Three-dimensional Single Large Object Placement Problem (SLOPP). As explored in the previous chapter, this variant concerns a set of weakly heterogeneous items (boxes) and a single larger object (container). The SLOPP is an output maximization problem which aims to maximize the value, in our case the volume, of the placed boxes. These boxes must be placed completely inside the container and they cannot overlap with each other.

Bortfeldt and Wäscher (2013) have provided a very thorough review of all sets of constraints currently considered in the literature of 3D Container Loading Problems, listing which constraints are reported in over 150 papers up until the year 2012. The most significant conclusion discussed concerns the lack of papers that consider an acceptable number of practical constraints, something which would make them more comparable to the problems found in the industry. In their survey regarding all the 3D packing problem categories, 74.9% of the publications consider at most two sets of practical constraints, with the majority being orientation and stability-related. At most seven sets of constraints were accommodated, featuring in only three papers, 1.8% of the publications, which are Bortfeldt and Gehring (1999), Egeblad, Garavelli, et al. (2010) and Olsson et al. (2020).

Bortfeldt and Gehring (1999) proposed a genetic algorithm to handle different combinations of sets of practical constraints to place regular items inside a single container, while Egeblad, Garavelli, et al. (2010) presented a solution approach

for the problem of loading a container with pieces of furniture (irregular items). More recently, Olsson et al. (2020) introduced a specific real-world problem where regular and irregular items are first packed into stacks and then placed in a container. Due to the unique problem specification associated with these three papers, each considers a different set of constraints.

These three papers address the 3D Single Knapsack Problem (SKP), a counterpart of the SLOPP and the most addressed container loading variant in the literature. The only difference between the SLOPP and the SKP is that while the former considers a set of weakly heterogeneous items to be placed in a single container, the latter consists of a set of strongly heterogeneous items. Although this difference might seem negligible, solution approaches which take into account the details of the items can generate higher quality solutions than those which treat all identically. With this in mind, we limit the scope of our research to container loading problems with weakly heterogeneous items.

Liu et al. (2011) proposed a problem variant with five sets of practical constraints, which is the largest number of sets of practical constraints considered in the literature for the SLOPP. They proposed a hybrid tabu search approach which considers orientation, stability, connectivity, weight limit and weight distribution constraints that are evaluated by solving real-world data. Other important papers from the literature concerning the SLOPP in which at most four sets of practical constraints accommodated are: Eley (2002), Hifi (2002), Bischoff (2006), Christensen and Rousøe (2009), Fanslau and Bortfeldt (2010), Ren et al. (2011), Junqueira et al. (2012), Moon and T. V. L. Nguyen (2014), Layeb et al. (2017) and Saraiva et al. (2019). Most recently Ramos et al. (2018) have introduced a more realistic type of weight distribution constraint, known as load balance constraints.

With this lack of approaches in the literature with a realistic set of practical constraints, we consider seven different sets of practical constraints: orientation, load balance, loading priorities, positioning, stacking, stability and weight limit. Therefore, our objective is to propose a well-performing matheuristic method which is able to solve problems with a considerable number of practical constraints and guarantee consistent results of good quality. The method consists of a wall-building approach composed of a greedy improvement heuristic and two mathematical formulations which divide between themselves the computational effort associated with considering all of the practical constraints. The method is structured in such a way that it is clear where each constraint is included and where extra constraints could be added.

To demonstrate the simplicity with which the method can be adapted to other problem variants, we also address the Single Stock-Size Cutting Stock Problem (SSSCSP), which is a multiple container variant of the SLOPP considering a

set of identical containers where the objective is to minimize the number of containers used to place all (weakly heterogeneous) items. The results from the computational experiments concerning both SLOPP and SSSCSP problems verify the robustness and flexibility of the approach, showing how it is capable of generating high quality solutions for the experimented instances when combining different sets of practical constraints.

The remainder of this chapter is structured as follows. Section 3.2 details the sets of practical constraints considered. A description of the solution method is provided in Section 3.3. Computational results evaluating the efficiency and adaptability of the method are presented in Section 3.4, while in Section 3.5, we detail its adaptation to the SSSCSP and validate its performance with instances from the literature. Finally, conclusions and future research directions are presented in Section 3.6.

3.2 Practical constraints

The practical constraints considered in this chapter conform to the definitions provided by Bortfeldt and Wäscher (2013):

Orientation constraints define how many orientations and which ones are allowed for each box type. Considering that all boxes are regular cuboids and their placement is orthogonal to the container axis, up to six orientations are allowed.

Stability constraints can be divided into two types: *vertical stability* guarantees each box has a minimum percentage of its base completely supported by either the container's floor or other boxes, while *horizontal stability* ensures that each box is supported on all four sides by either other boxes or the container's walls.

Stacking constraints guarantee that no box is damaged by carrying more than its maximum load (fragility). The most well-known fragility concept considered is the one whereby a box type cannot withstand any load placed atop it, allowing it to be only placed above other boxes in order to not be damaged.

Weight limit constraints limit the combined weight of the boxes placed inside the container to a predefined maximum.

Loading priorities constraints prioritize types of boxes that must have their full demand placed in the container.

Positioning constraints ensure that boxes of type t must be placed such that

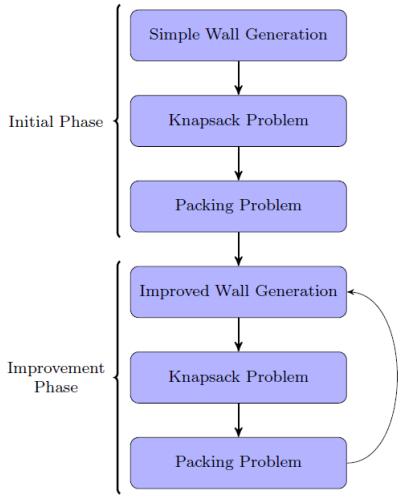
they respect a specific position relative to boxes of type s (*in front of*, *behind*, *to the left of*, or *to the right of*).

Load balance constraints are more precise than the weight distribution considered by Bortfeldt and Wäscher (2013). They are based on the characteristics of the vehicle transporting the container. Each vehicle has its own load distribution diagrams which define the feasibility domain for the location of its cargo's center of gravity. These diagrams are created in accordance with national, regional and continental legislation and they should detail: the maximum front axle load, rear axle load and gross vehicle weight; the minimum required steering axle load as well as the driving/traction axle load.

3.3 Solution method

The solution method consists of a wall-based approach with a greedy improvement phase. It consists of two parts: an initial phase which quickly generates a first solution of reasonable quality followed by an improvement phase which aims to improve the first solution within a given time limit. This method has a structure composed of modules that were created in order to facilitate the removal or addition of constraints, thereby accomodating a range of applications (see Figure 3.1).

Figure 3.1: Structure of the solution method.

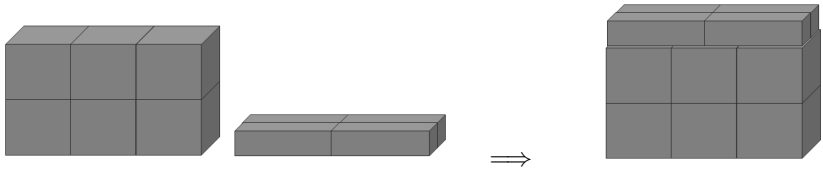


3.3.1 Wall generation

The wall generation is based on the method of Zhu and Lim (2012). A summary of the procedure is detailed in the following paragraphs and readers interested in the full details can consult the complete algorithm in the original paper.

The algorithm begins by generating structures containing boxes of the same type, referred to as homogeneous blocks, and iteratively placing homogeneous blocks of different types together in order to create more complex blocks. An illustrative example of how this works is shown in Figure 3.2. Walls are composed of any kind of box arrangements, starting from a homogeneous block with a single box until a combination of blocks. Due to this diversity, when considering the bounding box around each wall generated, only those which meet a certain occupancy percentage (*occperc*) are accepted, where *occperc* is defined as the volume of the boxes within the wall divided by the volume of the bounding box. These wall structures are created in such a way that all permitted rotations of the boxes are already considered and the boxes in each block are fully supported. This guarantees that the **orientation** and **vertical stability** constraints are respected throughout the entire solution method. For horizontal stability, we assume that fillings can be placed in the container in order to guarantee that each box has its sides supported by other boxes, container's walls or the inserted fillings. Fillings are pieces of materials used to fill the empty spaces inside the container. They can be made of foam, for example. Concerning the **load balance** constraints, in this step the walls are created already assuming them to be balanced along the Y-axis. Therefore, when the wall does not have 100% *occperc*, subsets of boxes are shifted towards the center of the Y-axis in order to keep the geometric centre of the walls in the middle of the container's width. The **stacking** constraints are considered during the creation of each wall: any wall that violates the stacking constraint is ignored.

Figure 3.2: Complex block generation.



(a) Homogeneous blocks. (b) Complex block.

The first phase of the solution method generates what we will refer to as Simple Walls (Figure 3.1). These structures consist of walls containing at most two homogeneous blocks placed side by side along the Y-axis or one above another.

Limiting the number of combinations of block types in this phase of the method is for speed purposes. Given a large-scale problem with more than 100 available boxes, generating a long list of good quality walls which combine an unlimited number of homogeneous blocks is very time consuming. At this stage of the solution method, the idea is to simply generate a fast initial solution of good quality that will be improved in the next stage of the method.

For the Improvement Phase, typically only a subset of boxes is considered to create new walls. Thus, more complex walls which contain more than two homogeneous blocks placed together are generated. Although considering a subset of boxes is less time consuming, the total number of walls (*nwalls*) and the quality of the complex walls (*occperc*) accepted are also important factors which directly influence how much time is spent in this stage. One final way in which Simple and Complex wall generation process can be accelerated is by running the entire procedure in parallel.

3.3.2 Mathematical formulation

Even when considering the simplest SLOPP with only non-overlapping and bounding constraints the mathematical formulations currently proposed in the literature struggle to solve medium-sized problems. The reasons for this are multiple, as reported in Chapter 2. Including additional constraints to these formulations will continue to hinder their performance, negatively impacting solution quality. Therefore, our strategy is to divide the remaining sets of practical constraints into two separate formulations: one based on the Knapsack Problem and another based on the Packing Problem. The division is made in such a way that the first formulation functions as a filter, reducing the number of walls created into a small set of promising high quality walls which are then used as input to the second, more complex formulation. Each constraint is considered only where necessary so as to ensure the feasibility of the final solution. To avoid the need for any post-processing checks, some constraints are included in both formulations.

Knapsack Problem

The first formulation solved is based on the classical Knapsack Problem. Given a list of generated walls (Simple or Complex), the objective is to maximize the volume of the boxes placed in the container. The two basic constraints considered specify that the sum of the length of the chosen walls cannot exceed the container's length and the demand for each box type must be met. The practical constraints included are **weight limit** and **loading priorities**. This changes

the formulation from a General Knapsack Problem to a Multidimensional Knapsack Problem, which was first described by Lorie and Savage (1955) and Markowitz and Manne (1957).

The input data and variables employed within the formulation are detailed in Table 3.1, while the problem formulation itself is presented by (3.1)-(3.6):

Table 3.1: Input data and variables.

Parameters:	
$\mathcal{I}$:	set of walls
$\mathcal{T}$:	set of box types
$\mathcal{T}'$:	set of box types obligatory to be placed
$\mathcal{D}_t$:	total demand of box type t
P_{max} :	maximum permissible container weight
L :	length of container
l_i :	length of wall i
v_i :	volume of wall i
m_i :	weight of wall i
d_{ti} :	number of boxes of type t in wall i
Variables:	
p_i :	integer variables indicating the number of times wall i is used

$$\max \quad \sum_{i \in \mathcal{I}} v_i \cdot p_i \tag{3.1}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{I}} l_i \cdot p_i \leq L, \tag{3.2}$$

$$\sum_{i \in \mathcal{I}} m_i \cdot p_i \leq P_{max}, \tag{3.3}$$

$$\sum_{i \in \mathcal{I}} d_{ti} \cdot p_i \leq \mathcal{D}_t, \qquad \forall\, t \in \mathcal{T}/\mathcal{T}', \tag{3.4}$$

$$\sum_{i \in \mathcal{I}} d_{ti} \cdot p_i = D_t, \qquad \forall\, t \in \mathcal{T}', \tag{3.5}$$

$$p_i \in \mathbb{Z}_+, \qquad \forall\, i \in \mathcal{I}. \tag{3.6}$$

Objective function (3.1) maximizes the total volume used in the container. Constraint (3.2) is the general knapsack constraint, which in this case ensures that the container length will be respected. Constraint (3.3) limits the maximum

weight in the container. Constraints (3.4) and (3.5) ensure that the maximum demand for all boxes is not exceeded. Moreover, Constraints (3.5) ensure that the loading priority constraint is met for each box type $t \in \mathcal{T}'$. Finally, Constraints (3.6) set the variables' domains.

The solution of this Knapsack Problem provides a good estimate of which walls will be placed in the container and how many copies of each wall will be used. It is only considered to be an estimate given that this solution does not consider all the practical constraints. Therefore, when including the remaining constraints the current solution is not necessarily feasible. However, should one only wish to solve the SLOPP while considering only weight limit and/or loading priorities constraints, then this solution is valid.

Packing Problem

The packing formulation, based on Chen et al. (1995), is mainly composed of the constraints that require information concerning the specific position of walls in the container: the **load balance** and **positioning** constraints. **Weight limit** and **loading priorities** are also included to ensure that the solution provided by this formulation is feasible for the entire problem considered within this chapter. Since it is assumed that it is not possible to place a wall neither beside nor above/below another wall, the only variable associated with the coordinates of the walls is for the X-axis, which corresponds to the length of the container. Therefore, this is a one-dimensional packing problem that aims to maximize the volume of the boxes placed. The basic constraints included guarantee that the chosen walls are positioned fully inside the container and do not overlap one another.

Figure 3.3: Truck measurements.

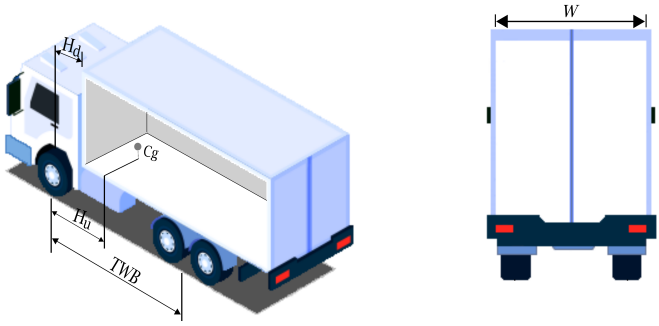


Figure 3.3 provides the necessary information from the truck in order to ensure that the load balance constraint is satisfied. The input data and supplementary variables employed within the formulation are detailed in Tables 3.2 and 3.3.

Table 3.2: Parameters.

C_g :	the truck's center of gravity
U :	weight of empty vehicle
TWB :	theoretical wheelbase - distance between front and rear horizontal center lines
H_d :	horizontal distance between the theoretical front-axle centre line and the back of the container
H_u :	horizontal distance between the theoretical front-axle centre line and the centre of gravity
FT_{max} :	maximum permissible front axle load
RT_{max} :	maximum permissible rear axle load
S_{min} :	minimum steering axle load
T_{min} :	minimum driving axle load
t_1, t_2 :	boxes of type 1 and 2 such that $t_1, t_2 \in \mathcal{T}, t_1 \neq t_2$

Table 3.3: Variables.

x_i :	continuous variable indicating the x-coordinate of wall i
s_i :	binary variable which is equal to 1 if wall i is placed in the container and 0 otherwise
g_i :	continuous variable indicating the location of the geometric centre on the X-axis of wall i in the container
a_{ij}, b_{ij} :	binary variables indicating the relative position of walls i and j (further from and closer to the container door, respectively)
P_{cargo} :	continuous variable which is equal to the total weight in the container
NC_x :	continuous variables indicating the numerator of the container's center of gravity on the X-axis (Equation 3.28)

The problem formulation is presented by (3.7)-(3.27):

$$\max \sum_{i \in \mathcal{I}} v_i \cdot s_i \tag{3.7}$$

$$\text{s.t. } x_i + l_i \leq x_j + (1 - a_{ij}) \cdot L, \qquad \forall\, i, j \in \mathcal{I}, i < j, \tag{3.8}$$

$$x_j + l_j \leq x_i + (1 - b_{ij}) \cdot L, \qquad \forall\, i, j \in \mathcal{I}, i < j, \tag{3.9}$$

$$a_{ij} + b_{ij} \geq s_i + s_j - 1, \qquad \forall\, i, j \in \mathcal{I}, i < j, \tag{3.10}$$

$$x_i + l_i \leq L + (1 - s_i) \cdot L, \qquad \forall\, i \in \mathcal{I}, \tag{3.11}$$

$$\sum_{i \in \mathcal{I}} d_{ti} \cdot s_i \leq \mathcal{D}_t, \qquad \forall\, t \in \mathcal{T}, \tag{3.12}$$

$$\sum_{i \in \mathcal{I}} d_{ti} \cdot s_i = D_t, \qquad \forall\, t \in \mathcal{T}', \tag{3.13}$$

$$\sum_{i \in \mathcal{I}} m_i \cdot s_i = P_{cargo}, \tag{3.14}$$

$$P_{cargo} \leq P_{max}, \tag{3.15}$$

$a_{ij} = 0$ if wall i contains box type t_1, wall j contains
box type t_2 and a positioning constraint between $\forall\, i, j \in \mathcal{I}, i < j,$
box types t_1 and t_2 is active, $\forall\, t_1, t_2 \in \mathcal{T}, t_1 \neq t_2,$ (3.16)

$b_{ij} = 0$ if wall i contains box type t_2, wall j contains
box type t_1 and a positioning constraint between $\forall\, i, j \in \mathcal{I}, i < j,$
box types t_1 and t_2 is active, $\forall\, t_1, t_2 \in \mathcal{T}, t_1 \neq t_2,$ (3.17)

$$g_i \leq L \cdot s_i, \qquad \forall\, i \in \mathcal{I}, \tag{3.18}$$

$$g_i \leq x_i + \frac{l_i}{2}, \qquad \forall\, i \in \mathcal{I}, \tag{3.19}$$

$$g_i \geq x_i + \frac{l_i}{2} + L \cdot s_i - L, \qquad \forall\, i \in \mathcal{I}, \tag{3.20}$$

$$\sum_{i \in \mathcal{I}} m_i \cdot g_i = NC_x, \tag{3.21}$$

$$NC_x \leq P_{cargo} \cdot (TWB - H_d) - TWB \cdot FT_{max} + U \cdot (TWB - H_u), \tag{3.22}$$

$$NC_x \leq TWB \cdot RT_{max} - U \cdot H_u - P_{cargo} * H_d, \tag{3.23}$$

$$NC_x \geq P_{cargo} \cdot (H_d - S_{min} \cdot TWB - TWB) + U \cdot (TWB - H_u - S_{min} \cdot TWB), \tag{3.24}$$

$$NC_x \geq U \cdot (T_{min} \cdot TWB - H_u) + P_{cargo} \cdot (T_{min} \cdot TWB - H_d), \tag{3.25}$$

$$s_i, a_{ij}, b_{ij} \in \{0, 1\}, \qquad \forall\, i, j \in \mathcal{I}, \tag{3.26}$$

$$x_i, g_i, P_{cargo}, NC_x \geq 0, \qquad \forall\, i \in \mathcal{I}. \tag{3.27}$$

Objective function (3.7) maximizes the total volume used in the container. Constraints (3.8)-(3.10) guarantee walls do not overlap with one another. Constraints (3.11) ensure that all the chosen walls are placed fully inside the container. Constraints (3.12)-(3.15) are similar to those in the knapsack formulation and ensure that the demand, loading priority and weight limit constraints are satisfied, respectively. Constraints (3.16)-(3.17) ensure the positioning constraints between two box types are fully respected. These constraints are currently used as an example of how such constraints should be modeled if walls containing box type t_1 must be placed in front of walls containing box type t_2.

Constraints (3.18)-(3.25) concern the load balance of the container. These constraints were adapted from E. Silva, Ramos, et al. (2018), the only paper in the literature that models this type of constraints. Constraints (3.18)-(3.21) calculate the geometric centre on the X-axis (NC_x), which combined with P_{cargo} determines the cargo's centre of gravity C_x on the X-axis:

$$C_x = \frac{NC_x}{P_{cargo}} \tag{3.28}$$

Since the walls are created in such a way that their center of gravity on the Y-axis is within the maximum permissible load difference between the left and right wheels, it is only necessary to guarantee the correct placement of the walls along the X-axis. Constraint (3.22) ensures the maximum permissible front axle load is respected and uses the equation of moments of forces of the rear axle and the container's total weight. Constraint (3.23) ensures the maximum permissible rear axle load is respected. Constraint (3.24) is responsible for the minimum permissible steering axle load, while Constraint (3.25) ensures that the minimum driving/traction axle load is respected. Finally, Constraints (3.26)-(3.27) set the variables' domains.

It is worth noting that if we only consider the walls present in the solution generated for the Knapsack Problem, the Packing Problem may result in a worse solution given that additional constraints must be considered in this formulation. To solve this issue, a more diverse set of walls is provided as input for this formulation. Therefore, in addition to the walls in the solution of the Knapsack Problem, an extra subset of walls (*extrawalls*) is provided. These walls are selected at random from the list of walls generated which were not included in the solution of the Knapsack Problem. The motivation behind selecting random walls rather than the most promising ones is that preliminary experiments showed a slight dominance for the former strategy.

3.3.3 Improvement phase

This phase consists of a greedy approach used to improve the initial solution (Algorithm 1). Given a solution, the idea is to initially remove the wall(s) with the lowest quality (line 6) and, considering the boxes that belong to the removed wall(s) together with those not considered in the solution (not placed in the container), to generate more complex walls with a greater diversity of boxes which will result in an improved solution (line 7). The next iteration consists of solving both the Knapsack and Packing problems considering the new list of walls generated and the walls not removed from the previous solution (lines 8 and 9, respectively).

When an improved solution is found, the problem is updated and the procedure restarts the improvement phase by removing the worst wall of this new best solution (lines 11 and 12). If no improved solution is found, the first and second worst wall in the solution are removed in the next iteration, providing a bigger subset of boxes for the Improved Wall Generation. While no improvement is found this procedure is repeated always removing the next worst wall together with the previous ones (line 14). The method ends when the time limit is reached (line 5).

3.4 Computational Results

Experiments were performed to evaluate the method considering multiple combinations of practical constraints. Instances BR originally proposed by Bischoff and Ratcliff (1995) are utilized for this purpose. The BR instance set consists of 15 classes (BR1 - BR15) with each class containing 100 instances. The first seven classes are characterized by weakly heterogeneous sets of boxes, while the remaining eight sets contain strongly heterogeneous sets of boxes. Since our goal is to solve SLOPP problems, we limit ourselves to instance classes BR1 to BR7. The sets gradually increase with respect to heterogeneity, meaning that BR1 contains the most homogeneous instances while BR7 contains the most heterogeneous instances.

More recently, E. Silva, Ramos, et al. (2018) and Ramos et al. (2018) recreated these instances to consider the boxes' weight. Further details concerning how these instances were updated are presented by Ramos et al. (2018). For our experiments, new information must be included in order to accommodate the additional constraints we consider. For example, the priority of a box to be placed in the container. Therefore, an extended set of BR Instances was

Algorithm 1: Solution method

Input: set of box types to be placed;
container dimensions and parameters (if load balance constraint is active);
Output: set of boxes placed in the container and their respective positions;
`// Initial Phase`
1 Generate *Simple Walls* considering all boxes from the input;
2 Solve the *Knapsack Problem* using the complete set of walls generated;
3 Solve the *Packing Problem* using the walls in the solution from line 2 and *extrawalls* chosen;
`// Improvement Phase`
4 n = 1;
5 **while** *time limit not reached* **do**
6 Remove the n worst wall(s) from the current best solution;
7 Generate *Improved Walls* with the removed boxes and the boxes not originally placed;
8 Solve the Knapsack Problem including the complete set of walls generated and the walls not removed in line 6;
9 Solve the Packing Problem including the walls in the solution of line 8, *extrawalls* and the walls not removed in line 6;
10 **if** *new best solution found* **then**
11 update current best solution;
12 n = 1;
13 **else**
14 n = n + 1;
15 **end**
16 **end**

generated which has been made publicly available online at `http://dx.doi.org/10.17632/vcp6btht2h.1`

Concerning the parameters for wall generation, the maximum number of walls accepted (*nwalls*) is 10,000 and these walls must have a quality (*occperc*) of at least 90%. As for the additional walls (*extrawalls*) that are included in the Packing model (see Section 3.3.2), 50 walls are randomly chosen from the list of walls generated. These walls can be placed at most twice in the container, meaning that there can be up to 100 extra walls in total. The values for these parameters were chosen after preliminary experiments which sought to determine values which provided enough diversity for high quality solutions, but which did not introduce so many variables as to negatively impact the performance of the mathematical model. Section 3.4.1 provides more information concerning these parameter values.

To evaluate the robustness of the method, experiments were conducted using

different sets of constraints, as detailed in Table 3.4. Test cases 1 and 2 have been considered in the literature with the same BR instances and therefore a comparison against the existing state of the art is available in Sections 3.4.2 and 3.4.3, respectively. The remaining experiments are presented in Section 3.4.4.

Table 3.4: Constraints considered in each set of experiments.

Test Case	Orientation	Stability	Weight Limit	Load Balance	Loading Priorities	Positioning	Stacking
1	✓	✓					
2	✓	✓	✓	✓			
3	✓	✓	✓	✓	✓		
4	✓	✓	✓	✓	✓		
5	✓	✓	✓	✓	✓	✓	✓

The computer used for the experiments is equipped with the following hardware: Intel Xeon CPU E5-2860 @2.50GHz with 24 cores and 64GB of RAM. Experiments were executed with a runtime limit of 900s and one run of each formulation, executed with IBM ILOG CPLEX 12.8, was limited to 60s. These runtime limits were derived from preliminary experiments. Wall generation was conducted in parallel, using all 24 available CPUs.

3.4.1 Sensitivity Analysis

Some preliminary experiments to decide which parameter values to use in our method were necessary. Although many combinations of parameter values were compared, to simplify the reading experience and focus on the most important details, we separated our experiments into two groups: one concerning the quality of the walls accepted and one for the number of extra walls to be considered.

To evaluate the impact of the quality of the walls to be accepted (*occperc*), experiments were run considering the first 10 instances of each instance set for test case 1 and test case 5, the two extremes in terms of number of constraints considered. Three different wall quality percentages were compared: 80%, 90% (our default value) and 99%. Figures 3.4 and 3.5 illustrate the performance of the two test cases considering these different values.

The first observation is that considering walls with 80% or 99% of quality the solutions obtained are always worse than those obtained with 90%. The results for $occperc = 99\%$ are a clear outlier, producing the worst results accross all instance sets. When considering additional constraints (test case 5) the gap between 80% and 90% becomes more pronounced. Therefore, given the results

Figure 3.4: Impact of the wall quality parameter for test case 1.

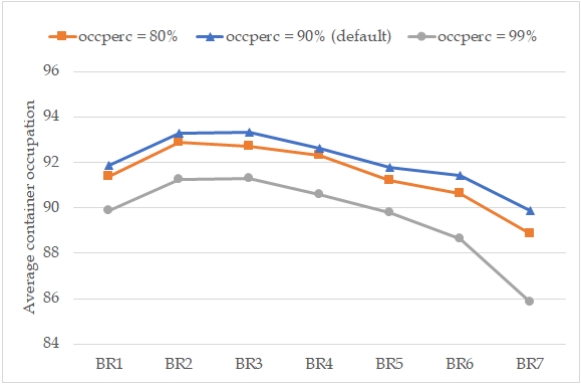
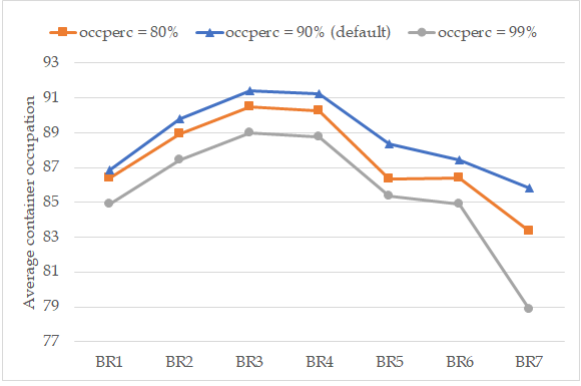


Figure 3.5: Impact of the wall quality parameter for test case 5.



obtained across all previous test cases and the similarities in the last test case, we decided to use $occperc = 90\%$.

For the second evaluation, the number of additional walls that should be included for the packing model, we consider test case 2 and test case 5. Test case 1 is discarded from these tests since it only considers orientation and stability constraints which, as already mentioned, does not require solving the Packing model and therefore no extra walls are ever considered. Following the same idea as before, two different number of extra walls were evaluated for the first 10 instances of each instance set: 50 and 100 different extra walls. Figures 3.6 and 3.7 illustrate the performance for these parameters.

After fixing the other parameter values, it is evident that considering 50 different

Figure 3.6: Impact of the number of extra walls parameter for test case 2.

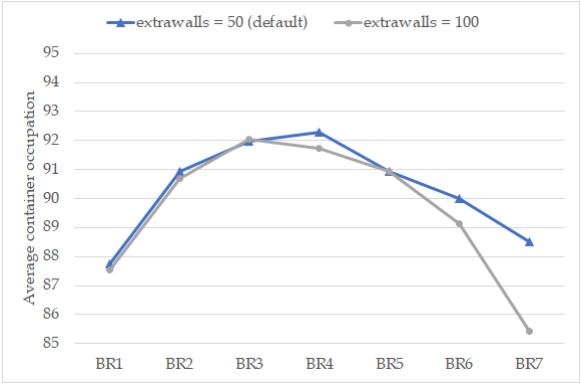
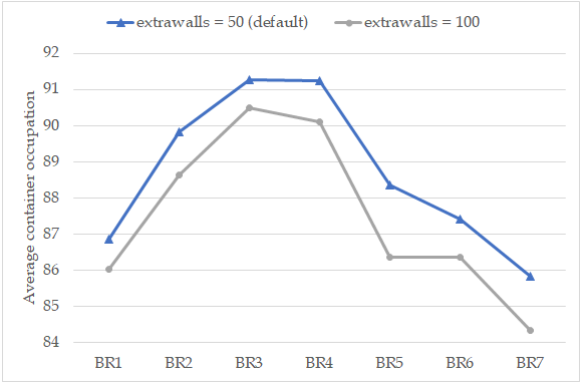


Figure 3.7: Impact of the number of extra walls parameter for test case 5.



extra walls performs best, particularly for test case 5. For test case 2, even though the results from BR3 and BR5 are, on average, better for *extrawalls* = 100 (0.20 and 0.10 better, respectively), better solutions are achieved for the remaining instance sets when considering *extrawalls* = 50. This is particularly evident in the last two instance sets, with BR6 and BR7 having an average solution difference of 0.78 and 3.11, respectively. Therefore, our selection concerning *extrawalls* was 50.

3.4.2 Test case 1

Considering only orientation and stability constraints, Araya, Guerrero, et al. (2017) produce the current best known results in the literature. They improved a

beam search solution method proposed by Araya and Riff (2014) for the specific packing problem which only considers these two constraints. Their improved method contains a new evaluation function with new evaluation criteria for ranking which boxes should be placed next. Liu et al. (2011), as already mentioned, presents the only SLOPP approach in the literature considering multiple (> 4) constraints being the most closely related approach compared to ours. Despite addressing multiple constraints, considering that not all those (connectivity and weight distribution) are similar to ours, the two handled in this section are the ones that can be properly analyzed.

We therefore compare our approach against the best results in the literature as well as the most closely related SLOPP approach. The performances of each method are presented in Table 3.5. The values correspond to the average container occupation when solving all 100 instances in each set.

Table 3.5: Orientation and stability constraints.

Instance set	Liu et al. (2011)	Our approach	Araya, Guerrero, et al. (2017)
BR1	88.14	92.64	94.74
BR2	89.52	92.88	95.38
BR3	90.53	93.05	95.65
BR4	90.75	92.54	95.48
BR5	90.79	92.13	95.33
BR6	90.74	91.85	95.23
BR7	90.07	90.89	94.66
Average	90.08	92.28	95.21

When comparing the approaches which consider a larger set of constraints, our matheuristic was able to obtain better average results across all instance classes, with an average improvement of 2.2pp (percentage point). When comparing our results against the best in the literature (Araya, Guerrero, et al. 2017), our approach has an average of 2.93pp less container occupation. As expected, the worst results obtained by our method are for the instance class with the most heterogeneous instances (BR7). In terms of execution time, Araya, Guerrero, et al. (2017)'s approach was limited to 150s. Liu et al. (2011) do not specify the execution time in their experiments. It is also important to mention that all published results employed different computer settings.

3.4.3 Test case 2

Ramos et al. (2018) are the only academics who consider orientation, stability, load balance and weight limit constraints when solving the SLOPP. They proposed a genetic algorithm to solve the Container Loading Problem considering these four constraints.

The characteristics of the vehicle are required to model load balance. Ramos et al. (2018) used the characteristics of the “Volvo Truck FE 62TR Air Ride" Trucks (2015), whose dimensions are compatible with the dimensions of the container used in the BR instances. Figure 3.3 and Table 3.6 detail the vehicle parameters. The performance of our method, compared against Ramos et al. (2018) is presented in Table 3.7.

Table 3.6: Container parameters.

Parameters	Values	Parameters	Values (ton-force)
H_d	0.832m	U	6.840
H_u	1.799m	P_{max}	19.160
TWB	4.397m	RT_{max}	19.000
T_{min}	25%	FT_{max}	7.100
S_{min}	25%		
LTR_{max}	10%		

Table 3.7: Orientation, stability, weight limit and load balance constraints.

Instance set	Our approach	Ramos et al. (2018)
BR1	90.08	91.31
BR2	90.83	92.40
BR3	91.85	93.32
BR4	91.56	93.58
BR5	91.33	93.60
BR6	90.67	93.48
BR7	89.64	93.35
Average	90.85	93.01

As expected, the results show that adding extra constraints, in general, results in a deterioration of solution quality. The average gap with respect to the state of the art heuristic method created specifically to solve the problem with this set of constraints is 2.16pp, and thus the proposed method maintains the same competitive behaviour exhibited in the previous experiments. In terms of execution time, again using different computer settings, the genetic algorithm, on average, was executed for at most 65s. As with the previous experiments, the worst results obtained by our approach were for the least homogeneous set of instances (BR7).

3.4.4 All test cases

The experiments in this section aim to show how consistent are the solutions when gradually introducing new constraints. Considering that results for the orientation, stability, weight limit and load balance have already been shown throughout the preceding sections, we begin by gradually including loading

priorities, stacking and, finally, positioning. This will complete the full set of constraints.

Given that the constraints included are directly related to box types, for experimentation purposes, some assumptions are required. When considering the loading priorities constraints, two test cases were created. The first (test case 3) assumes that the first half of all box types must have their full demand met. For example, if an instance has four box types, box types 1 and 2 must have their entire demand fulfilled. For test case 4, the second half of all box types must have their entire demand placed in the container, meaning that this time box types 3 and 4 must have their full demand fulfilled.

For test case 5, the positioning constraints also require some assumptions. Boxes of type 1 are to be delivered first and, therefore, they should be placed closer to the back of the truck than the other box types. The stacking constraints considered here use the concept of the most extreme fragility of a box. Therefore we assume that the last box type of each instance can only be placed above other boxes. For example, if an instance set contains five box types, then boxes of type 5 cannot be placed under any other box. Again, box type 1 must be delivered first and should therefore be placed closer to the back of the truck than the other box types. We also maintain the same loading priorities as in test case 3, that is, the first half of all box types must have their entire demand fulfilled. Table 3.8 details the average occupation of each instance class for each test case.

Table 3.8: Average container occupation for different combinations of constraints.

Instance set	Test case 1		Test case 2		Test case 3		Test case 4		Test case 5
BR1	92.64		90.08		89.48		89.17		89.24
BR2	92.88		90.83		89.90		89.19		89.44
BR3	93.05		91.85		90.52		90.75		90.71
BR4	92.54		91.56		90.88		90.43		90.44
BR5	92.13		91.33		90.79		90.30		90.22
BR6	91.85		90.67		89.68		89.99		88.03
BR7	90.89		89.64		86.91		86.42		86.67
Average (diff)	92.28	(1.43)	90.85	(1.11)	89.74	(0.28)	89.46	(0.21)	89.25

Comparing the simplest version of the problem (test case 1 - orientation and stability constraints) against the most complex one (test case 5), the average difference in solution quality is 3.03pp. If we compare the difference of gradually adding new sets of constraints (consecutive test cases), then the inclusion of weight limit and load balance constraints (test case 2) are those with the highest impact, having a total average impact of 1.43pp compared to test case 1. The impact of the load balance constraints was already expected given that in the

literature it is known that these are the constraints which have the biggest impact when it comes to reducing the convex hull of feasible solutions.

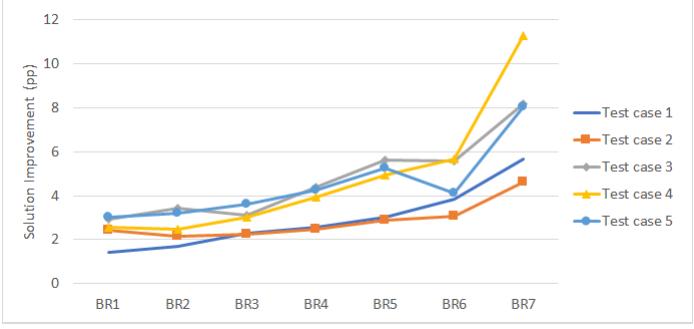
Comparing test cases 3 and 4, which differ from each other only by which box types have loading priorities, it can be observed that the former performs better than the latter. It is important to note that for BR3 and BR6, test case 4 actually performed slightly better (0.23pp and 0.31pp, respectively).

Test case 5, which combines all constraints considered in this chapter, performs worst, achieving an average occupation of 89.25%. Nevertheless, the fact that the method maintained average results of around 89% even after including all these constraints for the other three test cases demonstrates how robust our approach is.

In order to provide a thorough assessment of how our approach performs for all test cases, we also analyzed how much improvement there is from the initial solution compared to the final one, the average number of iterations performed in the method's final phase and how much time is spent in each phase.

Figure 3.8 illustrates the impact of the improvement phase. It can be observed that there is a direct correlation between the improvement percentage and the heterogeneity of instances. In other words, the most homogeneous instances (BR1) have the least improvement from initial to final solution while the most heterogeneous instances (BR7) have the biggest improvement. By analyzing the different test cases, it is also noticeable that those which consider more constraints benefit more from the destruction and recreation of walls in the improvement phase, with test case 4 having the largest average improvement.

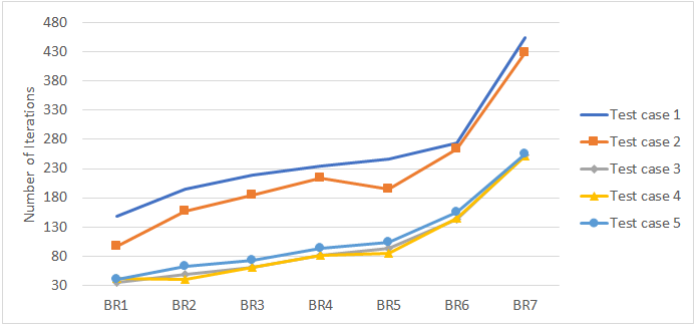
Figure 3.8: Average improvement percentage points from first to best solution found.



As for the average number of iterations, Figure 3.9 shows that there is a clear division between test cases 2 and 3. For all test cases, there is an increase in the

number of iterations as the instances become more heterogeneous, with BR1 having the smallest number of iterations and BR7 the greatest.

Figure 3.9: Average number of iterations.



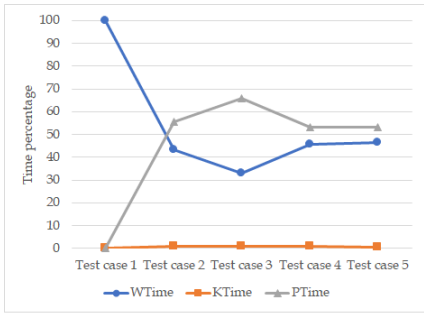
Finally, Figures 3.10 and 3.11 show how much time (%) is spent in each part of the solution method. *WTime*, *KTime* and *PTime* correspond to the percentage of the total execution time (900s) spent on the Wall Generation, Knapsack and Packing models, respectively.

The first observation is how the percentage of time spent solving the Knapsack model is very small ($< 1.5\%$). This indicates its efficiency as a "filter" for the walls to be considered in the Packing model. Since test case 1 only considers orientation and stability, and theses constraints are independent of the position of the boxes inside the container, the Packing model is not necessary, with almost 100% of its execution time spent in the wall building procedure. Test case 3 is the one that spends, on average, most of its execution time solving the Packing model. When analyzing the graphs from the perspective of each instance set, BR7 (Figure 3.11) is the one that demands more time spent in the Packing model while BR3 (Figure 3.10) demands the least average time in this same step of the solution approach.

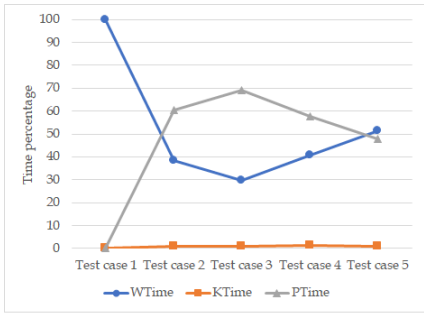
3.4.5 Results discussion

The main reason which explains why the results are worse than the state of the art is that for the proposed solution method to be able to handle different combinations of constraints, certain limitations must be imposed. For instance, the wall generation is implemented in such a way that it reduces its diversity, as described in Section 3.3.1. The reason for this, besides speeding up the matheuristic, is that when preliminary experiments were performed allowing more diversity, this improved the results of some sets of constraints while

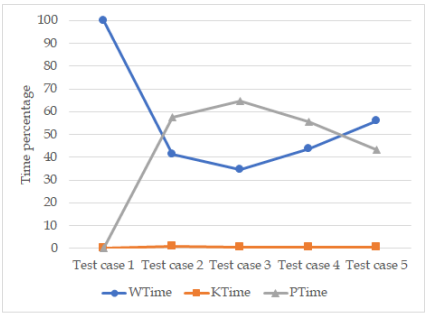
Figure 3.10: Time spent in each module of the solution approach (part I).



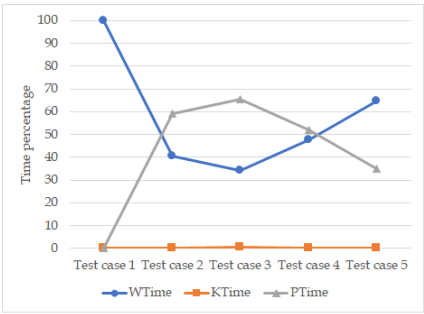
(a) Average time (%) spent per phase - BR1



(b) Average time (%) spent per phase - BR2



(c) Average time (%) spent per phase - BR3



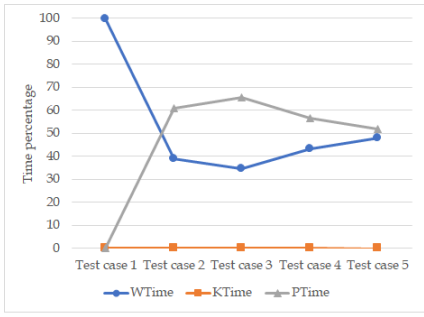
(d) Average time (%) spent per phase - BR4

simultaneously worsening others, creating a trade-off between the level of diversity and general solution quality.

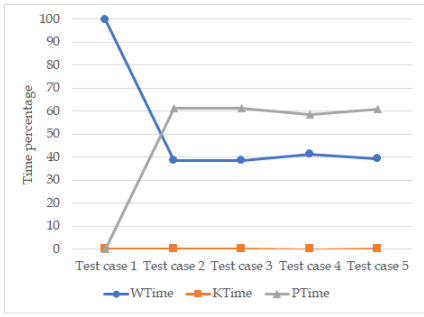
Therefore, the idea for the experiments was not necessarily to beat the state of the art in the literature, given that those results were achieved by specialized heuristics for each of the versions of the problem tested. Our goal with the preceding experiments is to demonstrate how a more flexible solution approach can guarantee consistent results of good quality. Thus, after analyzing all executions, the results indeed demonstrate the robustness of the solution method: when considering different combinations of constraints, high quality results are consistently achieved within the fixed time limit.

It is imperative to mention the importance of maintaining both the knapsack and packing models (when necessary), not only in terms of dividing the burden of the solution process, but to also facilitate the inclusion or removal of constraints, maintaining the hallmark of this approach. For the existing approaches in the

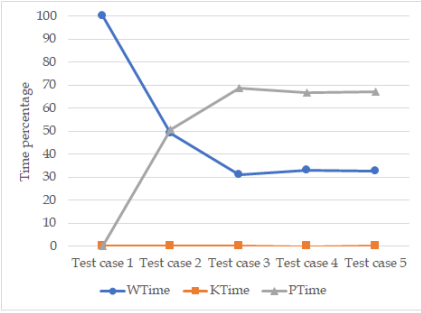
Figure 3.11: Time spent in each module of the solution approach (part II).



(a) Average time (%) spent per phase - BR5



(b) Average time (%) spent per phase - BR6



(c) Average time (%) spent per phase - BR7

literature, removing constraints would require changes in the solution procedure while the inclusion of additional constraints would require a redesign of the method or a post-processing procedure to accommodate the many infeasible solutions which would occur. To show how this approach can be easily modified to handle other problems, Section 3.5 adapts the proposed approach to consider multiple containers.

3.5 Multiple Container Approach

When adapting the single container solution approach to accommodate multiple containers, there are essentially two strategies for doing so: sequentially or simultaneously. The sequential method consists of reapplying the single container approach, one container at a time, and updating the demand for boxes after

each new container is full. Otherwise, the simultaneous method consists of defining a set of containers that is enough to fit all the boxes to be placed and then solving the problem considering this full set of containers. Our proposed approach is straightforward to adapt from a single container to a simultaneous multiple containers solution method.

As an equivalent to the SLOPP, we consider here the Single Stock Size Cutting Stock Problem (SSSCSP). This problem consists of a set of weakly heterogeneous boxes that must be placed inside a set of identical containers. It is an input minimization problem which aims to minimize the number of containers used.

To be able to solve a SSSCSP, changes need only be made to the mathematical models used and not something else. The main difference is the addition of an index in the variables which corresponds to the walls used. For example, for the SLOPP in the Knapsack formulation, variable p_i is an integer variable that defines how many times wall i is used. For the SSSCSP, this variable changes to p_{ij} which corresponds to how many times wall i is used in container j.

Applying the same logic as for the packing problem, variable s_i changes to s_{ij} which corresponds to wall i being used (or not) in container j. The same changes are applied to the variables that define the position of the wall inside the container (x_{ij}) and the position of the geometric center on the X-axis of the wall in the container (g_{ij} and NC_{x_j}). Furthermore, there is the addition of variable u_j in both formulations. This is a binary variable which is equal to 1 if container j is used and 0 otherwise.

In terms of constraints, aside from adapting all constraints to be enforced in each used container (by adding the extra index), it is necessary to include two sets of constraints in both models. The first set is a modification of the demand constraints: instead of limiting the number of boxes placed to be **at most** their respective demand, it is now mandatory that all the demand associated with each box type is fully met. Therefore, Constraints (3.5) and (3.13) are removed while (3.4) and (3.12) are replaced by (3.29) and (3.30), respectively:

$$\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} d_{ti}\cdot p_{ij} = \mathcal{D}_t, \qquad \forall\, t\in\mathcal{T}, \tag{3.29}$$

$$\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} d_{ti}\cdot s_{ij} = \mathcal{D}_t, \qquad \forall\, t\in\mathcal{T}. \tag{3.30}$$

The second modification concerns the inclusion of a set of constraints that guarantees that if a box is placed in a container then this container is considered as used ($u_j = 1$). These constraints are enforced by way of (3.31) and (3.32) in the Knapsack and the Packing formulations, respectively:

$$\sum_{i \in \mathcal{I}} p_{ij} \leq M \cdot u_j, \qquad \forall\, j \in \mathcal{J}, \tag{3.31}$$

$$\sum_{i \in \mathcal{I}} s_{ij} \leq M \cdot u_j, \qquad \forall\, j \in \mathcal{J}. \tag{3.32}$$

where M is a sufficient big number.

These constraints are used to control which containers are used in order to minimize each formulation's objective function:

$$\min \sum_{j \in \mathcal{J}} c_j \cdot u_j \tag{3.33}$$

where c_j is the cost of using container j.

Finally, one last constraint is included to reduce symmetry issues. Considering that all containers are identical, solutions in which the same items are grouped together and where only the order in which they are placed differs, are considered symmetrical solutions. Considering that all these solutions have the same objective value, it is always useful to remove some of them from the solution space in order to improve the performance of the solvers. We therefore include Constraints (3.34) in both formulations:

$$u_j \geq u_{j+1}, \qquad \forall\, j \in \mathcal{J}. \tag{3.34}$$

The complete formulations to solve the SSSCSP are provided in Appendix A. To evaluate the effectiveness of the adapted solution approach, experiments were executed with well-known instances from the literature. Ivancic et al. (1989) proposed 47 instances for the SSSCSP which vary the number of necessary containers from 2-55 (current best known results). Concerning the boxes, the instances contain 2-5 box types and the total number of boxes in each instance varies from 47-180. To evaluate our method, we compare its results against the state of the art in the literature, Zhu, Huang, et al. (2012), which consists of solving these instances considering only orientation and stability constraints.

The only difference between the experimental settings from the previous section is the time limit for the mathematical models. Since the formulations are bigger and more complex than those considering a single container, instead of limiting their execution time to 60s, we instead limit them to 180s. Table 3.9 details the instance solved, the obtained results by Zhu, Huang, et al. (2012) and the results obtained by our solution method considering the same constraints as

in the literature. The displayed values correspond to the number of containers used in the final solution.

The results demonstrate the same behavior as those obtained with the single container instances. When compared with the state of the art, the difference concerning the total number of containers used is only 3 (0.43%). The difference in a few solutions (three instances) happens for the same reason as in the previous experiments for the single container problem: the limitation concerning the diversity of walls.

Table 3.9: Number of containers used in the solution for each SSSCSP instance.

Instance	Zhu, Huang, et al. (2012)	Our approach
1	25	25
2	10	10
3	19	19
4	26	26
5	51	51
6	10	10
7	16	16
8	4	4
9	19	19
10	55	55
11	17	17
12	53	53
13	25	25
14	27	27
15	11	11
16	26	26
17	7	7
18	2	2
19	3	3
20	5	5
21	20	20
22	8	8
23	20	20
24	5	5
25	5	5
26	3	3
27	**4**	5
28	10	10
29	17	17
30	**22**	23
31	12	12
32	4	4
33	4	4
34	8	8
35	2	2
36	14	14
37	23	23
38	45	45
39	15	15
40	**8**	9
41	15	15
42	4	4
43	3	3
44	3	3
45	3	3
46	2	2
47	3	3
Total	693	696
Avg time (s)	117	900

3.6 Conclusions

By proposing a matheuristic framework to consider multiple practical constraints for the Single Container Loading Problem, more specifically for the Single Large Object Placement Problem (SLOPP), we were able to provide some solid academic foundations for realistic 3D regular packing problems. In total, up to seven sets of practical constraints are considered, a degree of generality which was previously missing in the literature. Newly generated instances from the literature considering different combinations of sets of constraints were used to evaluate the quality and robustness of the proposed method. All new instances have been made publicly available online in order to provide the necessary resources for future research.

Experimental results confirmed the quality of the method, which achieved high-quality solutions across all test cases. This demonstrates how efficient this general method can be, especially considering the fact that the best results in the literature are obtained by heuristics developed and calibrated for specific problems with a specific set of constraints. One could argue that another efficient means by which to arrive at high-quality solutions would be to execute all existing heuristics and pick the best solution obtained. However, this would prove problematic given that no efficient heuristic methods have been published for this problem considering this many constraints. Another problem would be that for each new constraint, new heuristic adaptations would be required. Further experiments demonstrated the robustness of the method by analyzing the effect of enforcing different combinations of constraints. The primary conclusion obtained from these results is that it is possible to take into account many different practical constraints in an efficient and simple way, bringing the problems solved in academia closer to those which occur in practice.

Another important contribution is that we were able to successfully structure a solution model with modules that can be easily modified, thereby facilitating the inclusion of additional constraints in a way which can benefit from advances in mathematical solvers. We also demonstrate how to extend this problem for the variant that considers multiple containers.

Future research directions include adapting the matheuristic to solve the remaining variants of container loading problems. Finally, another research proposal is to evaluate different possible structures used to organize boxes instead of walls, and better understand which structures are most suited to which contexts.

Chapter 4

Extracting maximal objects from three-dimensional solid materials

After successfully addressing a range of realistic regular 3D packing problems, more complicated cases involving irregular items are now investigated. The problem of extracting maximal objects from three-dimensional materials was specially selected for this thesis due to its many different practical applications such as circuit manufacturing, 3D printing and gemstone cutting.

The core of the problem involves positioning a small item inside a larger one. However, the key difference is that now this smaller item does not have a fixed size. Instead, the object to be placed contains a set of parameters which define its shape and the objective of this problem is to find the maximum volume for this item while staying completely inside the material. The problem can be subdivided into two variants depending on whether the solid material is convex or non-convex.

By making use of the characteristics of each problem variant, we propose a matheuristic approach for solving both problems. Mathematical formulations are also introduced in order to better ascertain the efficiency of the matheuristic. For the convex case, a non-linear programming model is proposed, while for the non-convex case a mixed integer linear programming model is developed.

4.1 Introduction

Maximizing the volume utilization rate of an object is a very common problem encountered throughout a wide variety of contexts, from robust product engineering, circuits manufacturing, to diamond and gemstone cutting. These problems typically consider a larger outer object (referred to as material throughout the chapter) within which a smaller object must be placed or from which it must be cut. Both material and objects can be irregular (convex or non-convex), with the non-convex case being the most challenging to solve.

The characteristics of these problems are strongly dependent upon the application which gives rise to them. Various strategies to solve these problems have been explored in the literature such as methods based on non-linear, linear, convex and semi-infinite programming. Moreover, it is often not even possible to redeploy methods designed for one application to another related application. For example, the gemstone cutting method (Winterfeld 2008) cannot be directly applied to diamond cutting (V. H. Nguyen and Strodiot 1992).

To maximize the volume of the object, both its orientation (rotation about all axes) and translation (position inside the material) must be considered. The objects are parametrized by their extreme points which are then triangulated in order to generate their bounding planes. In this chapter the goal is to address this very common 3D irregular problem: to maximize the volume of a convex or non-convex object inside a convex or non-convex raw material. We consider that the materials are all homogeneous and there is no specific direction in which they must be cut. The primary contribution is to consider general shapes and efficiently solve the problem by providing a robust solution method, finding good solutions regardless of the object and material. Therefore, we propose a matheuristic approach combining a linear programming (LP) formulation and a local search metaheuristic to solve both problem variants.

For the cases where the material is convex, a non-linear programming formulation (NLP) is proposed to compare the quality of the matheuristic for the small test instances where the object has at most 1400 vertices. When the material is non-convex, a cutting plane procedure is employed in the matheuristic approach. Additionally, we also propose a mixed-integer linear programming formulation (MILP) based on the tetrahedralization of the material to evaluate the matheuristic for small test cases with fixed orientations. Results validate the performance of the proposed matheuristic, which outperforms existing non-linear solvers and is able to solve instances that were too large for the proposed formulations to provide any solution. In order to establish benchmark instances for future research, all instances used in the experiments are available online.

The remainder of this chapter is organized as follows. Section 4.2 provides a

brief review of related studies in the literature. Section 4.3 and Section 4.4 describe the solution approaches and present computational experiments for when the material is convex and non-convex, respectively. Finally, Section 4.5 concludes and provides some guidance concerning future research directions.

4.2 Related work

Most of the three-dimensional packing papers deal with the regular (convex) case, in particular the container loading problem. Reviews concerning practical constraints, solution methods and mathematical models for the container loading problems have been provided by Bortfeldt and Wäscher (2013), Zhao et al. (2016) and E. F. Silva et al. (2019), respectively. By contrast, there have only been a few papers published concerning irregular 3D packing problems: Egeblad, Garavelli, et al. (2010) address irregular packing in relation to the container loading problem, while Romanova et al. (2018) and Egeblad, Nielsen, et al. (2007) deal with more general packing problems.

In the context of irregular 3D packing problems, we can also consider the problem of extracting a maximal object from a solid material. Given that the problem of extracting maximal objects from materials has been approached in different ways depending on the problem context, this section follows with a review of those two main problem contexts - design centering, diamond cutting and gemstone cutting - in addition to a review of the classical irregular 3D packing problem.

4.2.1 Design centering

The problem of design centering originated from the manufacturing of integrated circuits. The technologies used during the manufacturing of integrated circuits, particularly semicondutor integrated circuits, consist of components that are subject to statistical fluctuations. These fluctuations interfere with one another in such a way that, depending on the positioning of each circuit, the final product may not meet the required specifications. Therefore, the objective of this problem is to obtain a design that maximizes the number of circuits in accordance with its specifications.

Design centering can be modeled as a non-linear programming problem, where the objective is related to meeting the specifications demanded by a given circuit. These specifications serve as constraints which define the feasible region of the problem. A well-known approach to this problem involves approximating the

feasible region by way of a convex polyhedron. The objective is then to find the largest ellipsoid that fits inside this convex polyhedron. The center of the ellipsoid defines the design centering. This technique was first introduced by Director and Hachtel (1977) and referred to as simplicial approximation.

Different optimization models and methods have been developed using the simplicial approximation approach. Examples include linear programming (Abdel-Malek and Hassan 1991), quadratic programming (Wojciechowski and Vlach 1993), convex programming (Sapatnekar et al. 1994) and semi-definite programming (Hassan and Abdel-Naby 2012) models.

4.2.2 Diamond and gemstone cutting

A new class of problems which has recently been attracting academic attention concerns diamond and gemstone cutting. These problems are considered generalizations of the design centering problem and therefore similar techniques can be used to solve them.

The diamond cutting problem consists of finding the largest diamond of predefined shape within a given rough stone. V. H. Nguyen and Strodiot (1992) propose a global optimization algorithm for finding the maximum difference between two convex functions in order to extract the largest diamond of prescribed orientation.

Gemstone cutting, considered a generalization of diamond cutting, has a wider variety of shapes and does not require a prescribed number of faces (boundary planes). Winterfeld (2008) develop an interior-point method to solve gemstone cutting problems by formulating them as a general semi-infinite program. Maag (2015) also utilizes semi-infinite programming and applies a collision detection method together with a parametric optimization technique to fit a gemstone inside a rough stone.

Da Silva et al. (2012) develop a two-level heuristic approach to solve the gemstone cutting problem. The first of these levels finds the largest scaling factor for a given orientation and translation, while the second deploys a genetic algorithm to maximize the scaling factor over all orientations and translations. Küfer et al. (2015) goes one step further and considers different solution approaches for the maximal material usage in gemstone cutting when cutting one or more jewels from a single rough stone.

4.2.3 Irregular object cutting and packing in 3D Euclidean space

The problem of extracting maximal objects from 3D materials and the classical 3D irregular cutting and packing problem share the same geometrical concern: the non-overlapping constraint. In the first case, the object being extracted must not overlap with the material's outer region (boundary). The second case also has this constraint, however it must also ensure that there is no overlap between multiple objects being packed. Different geometric tools have been previously employed to deal with irregular problems which have been reviewed by Bennell and Oliveira (2008).

The 3D object packing literature contains several studies focusing on linear and non-linear mathematical formulations. Among them, Stoyan et al. (2005) introduce a linear mathematical formulation for packing convex polytopes with fixed orientations into a parallelepiped. Chernov et al. (2010) present a non-linear mathematical formulation based on phi-functions for 2D and 3D object packing. They also provide a general algorithmic approach using phi-functions to solve the packing problem. Romanova et al. (2018) introduce a non-linear programming formulation for packing concave polyhedra into a cuboid with continuous rotations and translations. They also develop a multi-start solution algorithm to search for better solutions. A recent review of mathematical models in the irregular object cutting and packing literature is available thanks to Leao et al. (2019).

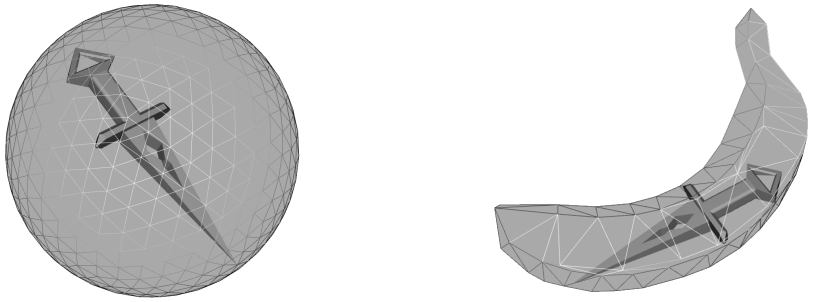
In addition to the aformentioned mathematical formulations, a few heuristic approaches have been published in the literature. A fast neighborhood search heuristic for 3D packing of irregular objects is derived from the 2D approach by Egeblad, Nielsen, et al. (2007). The algorithm utilizes a guided local search to escape from local optima. More recently, Ma et al. (2018) develop a hybrid algorithm for packing irregular objects inside an irregular 3D space. Their algorithm combines a continuous optimization procedure and a combinatorial swapping technique. The continuous optimization part is employed to obtain an initial packing configuration whereas the combinatorial swapping technique swaps the placement of objects to help escape local optima.

4.3 Convex outer object

To formally define the problem, we are given a solid material of fixed size and an object with predefined parameters, the objective is to extract the maximal object from inside the given material. Given an object with its initial parameter values

($scale = 1$), maximizing its volume corresponds to maximizing its scale, which is possible by means of translating (repositioning) and rotating the required object inside the material. Figure 4.1 shows how this problem can be subdivided into two variants depending on whether the material is convex or non-convex.

Figure 4.1: The two problem variants addressed in this chapter.



(a) Convex material. (b) Non-convex material.

This section focuses on the problem of **E**xtracting the **M**aximal **O**bject from a **C**onvex Material (EMOC). We initially describe the EMOC with the help of a non-linear mathematical programming formulation (NLP). This formulation is based on a rotation matrix defined by the Tait-Bryan angles, a form of the Euler angles (Euler 1776) commonly used in the fields of aircraft design, computer graphics and 3D printing.

4.3.1 Non-linear programming formulation

We define the parameters and decision variables necessary for formulating the EMOC as an NLP in Tables 4.1 and 4.2, respectively.

Table 4.1: Input data

$\mathcal{I}$	set of vertices of the object
v_{xi}, v_{yi}, v_{zi}	original coordinates of vertex $i \in \mathcal{I}$
$\mathcal{C}$	set of faces of the convex material
n_{xc}, n_{yc}, n_{zc}	components of the normal vector of face $c \in \mathcal{C}$
n_{dc}	equals to the product $(n_{xc}, n_{yc}, n_{zc}) \cdot (x_o, y_o, z_o)$, where (x_o, y_o, z_o) is any point on face $c \in \mathcal{C}$

Table 4.2: Decision variables

s	non-negative continuous variable indicating the scale of the object
t_x, t_y, t_z	continuous variables indicating the translation (placement) of the object along the X-, Y- and Z-axis, respectively
α, β, γ	rotation angles of the object

Equations (4.1) - (4.5) provide an NLP model for the EMOC:

$$NLP \quad \max \; s \tag{4.1}$$

$$\begin{aligned}
\text{s.t.} \; & n_{xc}(t_x + sv_{xi}\cos\alpha\cos\beta \\
& \quad + sv_{yi}(\cos\alpha\sin\beta\sin\gamma - \cos\gamma\sin\alpha) \\
& \quad + sv_{zi}(\sin\alpha\sin\gamma + \cos\alpha\cos\gamma\sin\beta)) \\
& + n_{yc}(t_y + sv_{xi}\cos\beta\sin\alpha \\
& \quad + sv_{yi}(\cos\alpha\cos\gamma + \sin\alpha\sin\beta\sin\gamma) \\
& \quad + sv_{zi}(\cos\gamma\sin\alpha\sin\beta - \cos\alpha\sin\gamma) \\
& + n_{zc}(t_z + sv_{xi}(-\sin\beta) \\
& \quad + sv_{yi}\cos\beta\sin\gamma \\
& \quad + sv_{zi}\cos\beta\cos\gamma) \\
& + n_{dc} \leq 0, \qquad \forall\, i \in \mathcal{I}, \forall c \in \mathcal{C},
\end{aligned} \tag{4.2}$$

$$s \geq 0, \tag{4.3}$$

$$t_x, t_y, t_z \in \mathbb{R}, \tag{4.4}$$

$$\alpha, \beta, \gamma \in \mathbb{R}. \tag{4.5}$$

Objective function (4.1) maximizes the scale of the object. Constraints (4.2) ensure that each vertex of the object is on the opposite side of the normal vector of every face of the material, thereby ensuring that each vertex of the object lies inside the boundary of the material. Constraints (4.3)-(4.5) define the domain of the variables.

4.3.2 Matheuristic approach for the EMOC

When the orientation of the object is fixed, in other words (α, β, γ) is set to some $(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$, the EMOC can be reduced to the following linear programming

(LP) formulation:

$$
\begin{aligned}
LP(\tilde{\alpha},\tilde{\beta},\tilde{\gamma}) \qquad & \max \ (4.1) \\
& \text{s.t.} \ \ n_{xc}t_x + n_{yc}t_y + n_{zc}t_z + \\
& \qquad sv_{xi}\bar{\alpha} + sv_{yi}\bar{\beta} + sv_{zi}\bar{\gamma} + n_{dc} \leq 0, \qquad \forall\, i \in \mathcal{I}, \forall c \in \mathcal{C}, \qquad (4.6) \\
& \qquad (4.3), (4.4),
\end{aligned}
$$

where

$$\bar{\alpha} = n_{xc}\cos\tilde{\alpha}\cos\tilde{\beta} + n_{yc}\cos\tilde{\beta}\sin\tilde{\alpha} + n_{zc}(-\sin\tilde{\beta}),$$

$$\bar{\beta} = n_{xc}(\cos\tilde{\alpha}\sin\tilde{\beta}\sin\tilde{\gamma} - \cos\tilde{\gamma}\sin\tilde{\alpha}) + (\cos\tilde{\alpha}\cos\tilde{\gamma} + \sin\tilde{\alpha}\sin\tilde{\beta}\sin\tilde{\gamma}) + n_{zc}\cos\tilde{\beta}\sin\tilde{\gamma},$$

$$\bar{\gamma} = n_{xc}(\sin\tilde{\alpha}\sin\tilde{\gamma} + \cos\tilde{\alpha}\cos\tilde{\gamma}\sin\tilde{\beta}) + n_{yc}(\cos\tilde{\gamma}\sin\tilde{\alpha}\sin\tilde{\beta} - \cos\tilde{\alpha}\sin\tilde{\gamma}) + n_{zc}\cos\tilde{\beta}\cos\tilde{\gamma}.$$

Constraints (4.6) ensure that all vertices of the object are inside the material.

The matheuristic method solves the LP for different orientations $(\tilde{\alpha},\tilde{\beta},\tilde{\gamma})$ of the object obtained via a local search framework, the Iterated Local Search (ILS), originally proposed by Lourenço and Martin (2003). The pseudo-code of the method is provided by way of Algorithm 2.

Algorithm 2: Matheuristic for the EMOC

Input: Solid material;
Object;
Initial solution (initSol);
Output: Maximal Object (bestSol);
Data: perturbations = 0;

```
currentSol = localSearch(initSol);
bestSol = currentSol;
while perturbations < maxPerturbations do
    currentSol = perturb(bestSol);
    currentSol = localSearch(currentSol);
    if currentSol > bestSol then
        bestSol = currentSol;
    end
    perturbations = perturbations + 1;
end
```

Algorithm 2 begins with an initial solution where the object has a very small volume (0.001) and is located in the center of the solid material with its default

orientation. *localSearch* is then called (line 1) to find the first improving solution (Algorithm 3). Following this, the main strategy of Algorithm 2 begins: as long as the maximum number of perturbations (*maxPerturbations*) is not reached (line 3), the algorithm repeats two procedures: *perturb* (line 4) and *localSearch* (line 5).

Two operators are applied in the *localSearch* procedure (Algorithm 3): *change* and *flip*. The *change* procedure (line 6) involves randomly modifying an angle ($\tilde{\alpha}$, $\tilde{\beta}$ or $\tilde{\gamma}$) within a given *range*, with the new solution calculated by solving $LP(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$. This *range* varies depending on the number of times (*runs*) this move will be called (line 5). *change* is ultimately executed *runs* times for each orientation (line 3). *flip*, also applied individually for each orientation, works by simply receiving one of the given angles, multiplying it by -1 and calculating the new solution (line 15). These two procedures are repeated until no solution improvements are found (line 1).

The procedure *perturb* consists of applying a perturbation to the current best solution in an attempt to avoid local optima. It consists of randomly modifying one of the angles from the current best solution and solving the $LP(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$ for the new orientation. This random value is chosen in such a way that it is always greater than the *range* used in the *localSearch* procedure, as otherwise *perturb* would be equivalent to *change*. The choice to modify only one of the three angles together with all the other parameters (*maxPerturbation*, *range*, *runs*) was based on the results from preliminary experiments.

4.3.3 Computational study

In this set of experiments we compare our matheuristic approach against the NLP model. The NLP models are generated using AMPL version 20191031 and solved with the following well-known NLP solvers: IPOPT 3.12.8, KNITRO 12.0 and CONOPT 3.17A.

As already mentioned, some of the matheuristic's parameters were defined based on preliminary experiments: the total number of perturbations (*maxPerturbations*) is set to 100, the interval within which the angles are modified (*range*) is 100 degrees ($\{\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}\} \pm 5$) divided into ten consecutive disjoint intervals (*runs*). This means that the first run considers changes from -5 to 5 degrees, the second run considers changes from 5 to 15 degrees and so on until the final run which considers changes from 85 to 95 degrees. Section 4.3.3 provides additional information concerning the option of modifying only one angle in the *perturb* procedure. The LP model is solved using Gurobi 8.1. The computer used for all experiments is equipped with the following hardware: Intel Core i7-2600 @3.40GHz with four cores and 8GB of RAM.

Algorithm 3: *localSearch* method

```
Input: Current solution (currentSol);
Output: Best solution found (bestSol);
Data: improvement = true;
bestSol = currentSol;
1  while improvement = true do
2      improvement = false;
3      for orientation in α̃, β̃ or γ̃ do
4          iter = 0;
5          while iter < runs do
               // Change the value of α̃, β̃ or γ̃ within a range
6              currentSol = change(bestSol, orientation, iter, range);
7              if currentSol > bestSol then
8                  improvement = true;
9                  bestSol = currentSol;
10             end
11             iter = iter + 1;
12         end
13     end
14     for orientation in α̃, β̃ or γ̃ do
           // Multiply the value of α̃, β̃ or γ̃ by -1
15         currentSol = flip(bestSol, orientation);
16         if currentSol > bestSol then
17             improvement = true;
18             bestSol = currentSol;
19         end
20     end
21 end
```

Considering that the NLP solvers are unable to produce good quality results for large-scale instances, we conduct experiments with materials that contain a small number of vertices such as *convex stone-*, *dodecahedron-*, *frustum-*, *sphere-* and *tetrahedron*-shaped materials (70, 32, 74, 642 and 4 vertices, respectively). As the objects may be of any shape, we also include irregular objects such as *apple*, *banana*, *dagger* and a *torus*-shaped object in our experiments. The objects used for all experiments are illustrated in Appendix B.

Sensitivity analysis

Considering that the perturbation step is one of the most important elements of the ILS, it is interesting to investigate the impact of introducing more randomness to this procedure as well as elaborating on our decision to modify only one of the orientations. Therefore, we compared three versions of our proposed approach. The first, ILS1, randomly modifies one of the angles of the current best solution in the *perturb* procedure. The second, ILS2, randomly modifies two angles of the current best solution. Finally, ILS3 randomly modifies all three angles simultaneously.

Experiments enable to compare the performance of these three approaches. Table 4.3 provides the average results (vol) of ten runs of each instance, their respective standard deviation ($std\ dev$) and average runtime $t(s)$.

The results demonstrate the superiority of the strategy ILS1, which generates the best average solutions values for all instances except *dodecahedron* x *apple*. As for ILS2 and ILS3, although both are unable to reach the best average solutions values for the same three instances, their solution values indicate that ILS3 is the worst performing strategy. This shows that applying randomness is only effective until a certain threshold which, as demonstrated, should be evaluated with preliminary experiments.

Another way in which ILS1 can be confirmed as the best strategy concerns the standard deviation. For ILS1, there is only one instance (*convex stone* x *banana*) where the solution value varied accross the different runs. For the other strategies, multiple instances exhibit some variation. In terms of time, all three strategies had a similar performance, with a negligible difference between them. Thus, after evaluating all these different criteria, it became clear that ILS1 represents the best choice.

Table 4.3: Comparison of the different strategies used in the *perturb* procedure.

Items		**ILS1**			**ILS2**			**ILS3**		
Material	Object	vol	std dev	t(s)	vol	std dev	t(s)	vol	std dev	t(s)
convex stone	apple	**359.62**	0.00	7.9	**359.62**	0.00	7.8	**359.62**	0.00	8.0
convex stone	banana	**419.41**	4.98	8.0	409.35	4.98	7.8	404.37	15.04	8.1
convex stone	dagger	**73.48**	0.00	15.7	**73.48**	0.00	15.8	**73.48**	0.00	17.2
convex stone	helmet	**341.77**	0.00	110.5	332.30	5.39	112.8	330.05	6.33	114.1
convex stone	star	**77.72**	0.00	6.2	**77.72**	0.00	6.2	**77.72**	0.00	6.4
convex stone	torus	**504.61**	0.00	13.0	**504.61**	0.00	13.5	**504.61**	0.00	13.9
convex stone	tree	**137.31**	0.00	9.1	**137.31**	0.00	8.9	**137.31**	0.00	9.2
dodecahedron	apple	135.94	0.00	4.9	**141.78**	0.00	4.9	**141.78**	0.00	5.1
dodecahedron	banana	**22.11**	0.00	4.9	**22.11**	0.00	4.8	**22.11**	0.00	5.1
dodecahedron	dagger	**3.03**	0.00	11.8	**3.03**	0.00	12.7	**3.03**	0.00	13.1
dodecahedron	helmet	**103.83**	0.00	101.0	**103.83**	0.00	105.3	**103.83**	0.00	108.8
dodecahedron	star	**8.56**	0.00	3.6	**8.56**	0.00	3.6	**8.56**	0.00	3.8
dodecahedron	torus	**52.73**	0.00	9.9	**52.73**	0.00	10.0	**52.73**	0.00	10.1
dodecahedron	tree	**45.44**	0.00	6.0	**45.44**	0.00	6.1	**45.44**	0.00	6.3
frustum	apple	**10.41**	0.00	7.8	**10.41**	0.00	8.0	**10.41**	0.00	8.1
frustum	banana	**2.36**	0.00	8.2	**2.36**	0.00	8.1	**2.36**	0.00	8.3
frustum	dagger	**0.31**	0.00	15.7	**0.31**	0.00	16.3	**0.31**	0.00	17.0
frustum	helmet	**16.25**	0.00	108.0	15.95	0.30	115.7	15.36	1.44	112.8
frustum	star	**0.77**	0.00	6.5	**0.77**	0.00	6.6	**0.77**	0.00	6.8
frustum	torus	**5.33**	0.00	13.5	**5.33**	0.00	13.7	**5.33**	0.00	14.0
frustum	tree	**3.94**	0.00	9.3	**3.94**	0.00	9.2	**3.94**	0.00	9.5
sphere	apple	**1.95**	0.00	45.8	**1.95**	0.00	45.9	**1.95**	0.00	48.6
sphere	banana	**0.28**	0.00	47.0	**0.28**	0.00	45.9	**0.28**	0.00	49.3
sphere	dagger	**0.03**	0.00	62.7	**0.03**	0.00	66.3	**0.03**	0.00	70.1
sphere	helmet	**1.66**	0.00	204.1	**1.66**	0.00	209.5	**1.66**	0.00	207.8
sphere	star	**0.10**	0.00	39.6	**0.10**	0.00	39.8	**0.10**	0.00	42.5
sphere	torus	**0.87**	0.00	61.1	**0.87**	0.00	61.5	**0.87**	0.00	64.8
sphere	tree	**0.64**	0.00	49.7	**0.64**	0.00	49.9	**0.64**	0.00	52.5
tetrahedron	apple	**0.76**	0.00	2.6	**0.76**	0.00	2.6	**0.76**	0.00	2.7
tetrahedron	banana	**0.35**	0.00	2.6	**0.35**	0.00	2.6	**0.35**	0.00	2.6
tetrahedron	dagger	**0.06**	0.00	9.4	**0.06**	0.00	9.8	**0.06**	0.00	9.8
tetrahedron	helmet	**1.29**	0.00	97.6	**1.29**	0.00	101.5	**1.29**	0.00	100.1
tetrahedron	star	**0.07**	0.00	1.4	**0.07**	0.00	1.4	**0.07**	0.00	1.4
tetrahedron	torus	**0.38**	0.00	6.8	**0.38**	0.00	7.2	**0.38**	0.00	7.3
tetrahedron	tree	**0.35**	0.00	3.6	**0.35**	0.00	3.7	**0.35**	0.00	3.8
	Average	**66.68**	**0.14**	**31.9**	66.28	0.30	32.7	66.05	0.65	33.4

EMOC results

Table 4.4 presents the results obtained by the NLP solvers and our proposed matheuristic, where *vol* corresponds to the volume of the object in the solution and $t(s)$ is the execution time in seconds. For the matheuristic approach, *vol* and $t(s)$ correspond to the average values obtained over ten runs, while *std dev* corresponds to the standard deviation of these results.

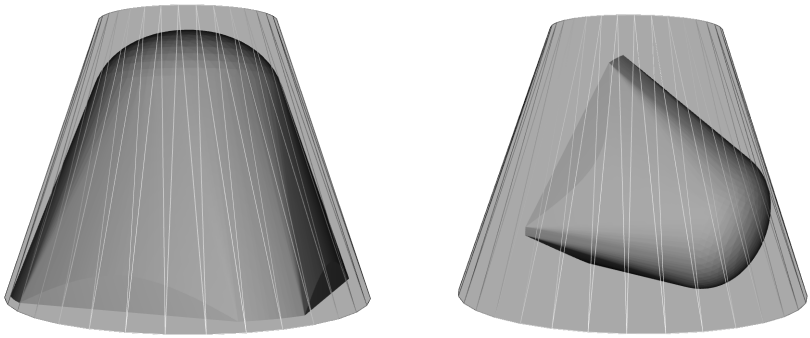
The results in Table 4.4 show the efficiency of the matheuristic, which obtains the best solutions for 32 out of the 35 instances and the best average results across all instances. The second instance (*convex stone* x *banana*) was the only instance for which different results were produced across the ten runs, resulting in a standard deviation of 4.98. For this instance, the maximum value found was 424.51 (gap of 1% from the best result found) and the minimum value was 414.36 (gap of 3.5% from the best result found). When comparing the NLP solvers, IPOPT obtained the best results, with an average gap of 0.8% from our results. Despite this gap of less than 1%, in terms of execution time, on average the matheuristic required less than half the time required by IPOPT.

Although all non-linear solvers obtained average results of high quality, it is important to highlight their limitations. When extracting the maximal *helmet* from a *sphere*-shaped material, the NLP became so large that AMPL was unable to process the problem on the computer used. By contrast, the matheuristic confirmed its robustness by solving the same instance with an average of 204.1 seconds. To maintain a fair comparison, the average results presented do not consider this instance. Figures 4.2 and 4.3 provide two examples of how the solutions produced by the matheuristic and IPOPT differ from each other. Figure 4.2 displays the solutions obtained by the proposed matheuristic (16.25) and the IPOPT solver (8.20) for the instance *frustum* x *helmet*. Meanwhile, Figure 4.3 illustrates the a case which the solution found by the matheuristic approach for the instance *convex stone* x *banana* is worse than that obtained by the IPOPT solver.

Table 4.4: NLP and matheuristic results when solving the EMOC.

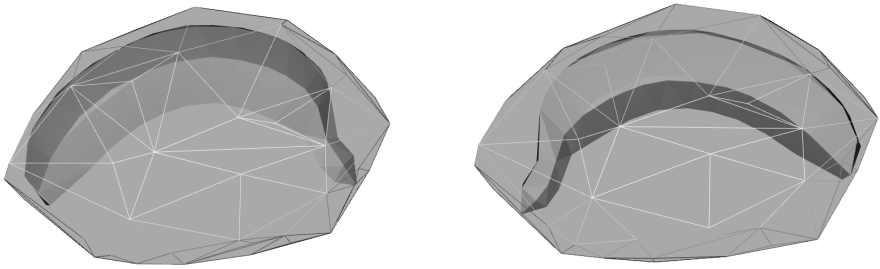
Items		**IPOPT**		**KNITRO**		**CONOPT**		**Matheuristic**		
Material	Object	vol	t(s)	vol	t(s)	vol	t(s)	vol	std dev	t(s)
convex stone	apple	358.89	25.0	357.16	4.6	355.43	2.9	**359.62**	0.00	7.9
convex stone	banana	**428.92**	17.1	374.96	3.0	**428.92**	8.8	409.35	4.98	8.0
convex stone	dagger	**73.48**	64.4	**73.48**	28.9	72.43	29.6	**73.48**	0.00	15.7
convex stone	helmet	335.44	191.1	**341.77**	226.9	335.44	8.2	**341.77**	0.00	110.5
convex stone	star	**77.72**	1.8	**77.72**	1.7	**77.72**	1.3	**77.72**	0.00	6.2
convex stone	torus	**504.61**	66.3	**504.61**	31.2	**504.61**	21.2	**504.61**	0.00	13.0
convex stone	tree	133.24	20.1	131.67	8.7	119.56	2.0	**137.31**	0.00	9.1
dodecahedron	apple	**141.78**	4.1	134.45	1.8	**141.78**	1.1	135.94	0.00	4.9
dodecahedron	banana	20.98	4.8	21.36	2.6	**22.11**	0.8	**22.11**	0.00	4.95
dodecahedron	dagger	3.02	18.6	3.02	3.5	3.02	2.6	**3.03**	0.00	11.8
dodecahedron	helmet	**103.83**	84.0	**103.83**	44.0	102.73	12.5	**103.83**	0.00	101.0
dodecahedron	star	8.53	1.0	8.53	0.2	8.53	0.3	**8.56**	0.00	3.6
dodecahedron	torus	40.92	8.1	**52.73**	3.9	**52.73**	3.2	**52.73**	0.00	9.9
dodecahedron	tree	44.08	8.3	44.08	1.5	44.08	0.6	**45.44**	0.00	5.8
frustum	apple	10.33	12.2	10.17	2.3	10.33	1.5	**10.41**	0.00	7.8
frustum	banana	2.05	9.7	2.16	2.7	2.10	2.0	**2.36**	0.00	8.2
frustum	dagger	**0.31**	27.5	**0.31**	12.7	0.25	3.1	**0.31**	0.00	15.7
frustum	helmet	8.20	179.9	7.91	196.7	8.20	29.8	**16.25**	0.00	108.0
frustum	star	**0.77**	3.0	0.62	0.3	0.46	0.1	**0.77**	0.00	6.5
frustum	torus	**5.33**	140.8	5.32	8.0	2.64	2.6	**5.33**	0.00	13.5
frustum	tree	3.85	18.8	2.84	6.2	3.86	2.3	**3.94**	0.00	9.3
sphere	apple	1.94	117.6	1.94	48.5	1.94	15.5	**1.95**	0.00	45.8
sphere	banana	**0.28**	317.3	**0.28**	93.0	**0.28**	15.2	**0.28**	0.00	47.0
sphere	dagger	**0.03**	271.1	**0.03**	332.6	**0.03**	55.9	**0.03**	0.00	62.7
sphere	helmet	-	-	-	-	-	-	**1.66**	0.00	204.1
sphere	star	0.09	13.3	0.09	8.4	0.09	1.6	**0.10**	0.00	39.6
sphere	torus	**0.87**	181.3	**0.87**	233.2	0.86	94.1	**0.87**	0.00	61.1
sphere	tree	**0.64**	155.1	**0.64**	83.4	**0.64**	11.8	**0.64**	0.00	49.7
tetrahedron	apple	**0.76**	0.1	**0.76**	0.1	**0.76**	0.1	**0.76**	0.00	2.6
tetrahedron	banana	**0.35**	0.1	**0.35**	0.1	**0.35**	0.1	**0.35**	0.00	2.6
tetrahedron	dagger	**0.06**	1.0	**0.06**	0.3	**0.06**	0.2	**0.06**	0.00	9.4
tetrahedron	helmet	**1.29**	4.1	**1.29**	4.2	0.44	0.1	1.28	0.00	97.6
tetrahedron	star	0.06	0.1	0.06	0.1	0.06	0.1	**0.07**	0.00	1.4
tetrahedron	torus	0.27	9.8	0.29	0.1	0.27	0.1	**0.38**	0.00	6.8
tetrahedron	tree	**0.35**	0.3	**0.35**	0.1	**0.35**	0.1	**0.35**	0.00	3.6
	Average	68.04	58.2	66.64	41.1	67.74	9.8	**68.59**	0.15	26.8

Figure 4.2: The helmets extracted from a *frustum*-shaped material (matheuristic x best NLP solution).



(a) Matheuristic solution (16.25). (b) IPOPT solution (8.20).

Figure 4.3: The bananas extracted from a *convex stone*-shaped material (matheuristic x best NLP solution).



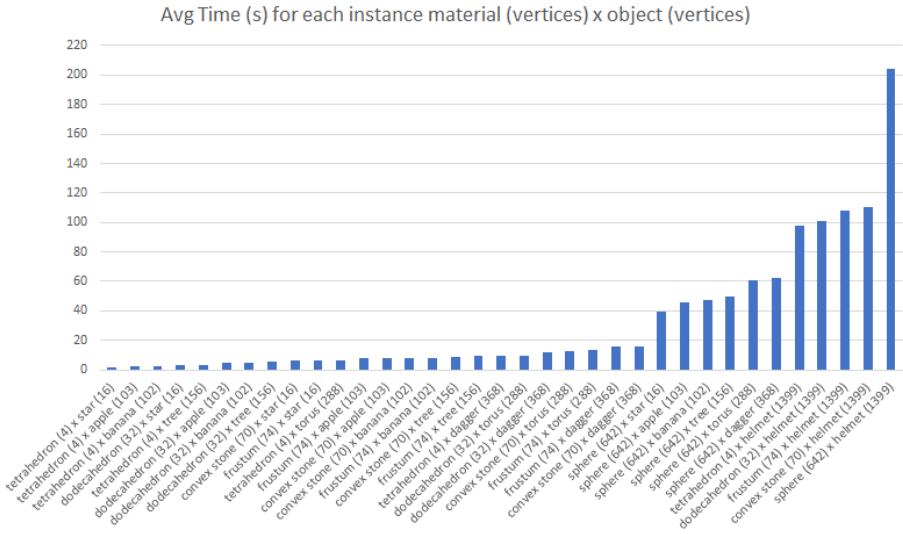
(a) Matheuristic solution (419.41). (b) IPOPT solution (428.92).

To assess the impact of instance sizes upon our solution approach, Figure 4.4 displays the (ordered) average execution time when solving each instance. The majority of instances require less than 20s to be solved. For the instances in this category, the last two consider objects with at most 368 vertices (*dagger*). Upon examining those instances which fall outside the aforementioned group, it is clear that there are boundaries in terms of number of vertices of the material and/or the objects considered which corresponds to instance's difficulty.

We can separate these remaining instances into two groups. The intermediate group, from 40s to 65s, include all the instances with sphere-shaped materials (642 vertices), with the exception of the instance *sphere* x *helmet* which considers the instance with the greatest total number of vertices and, therefore, occupies

the final group. This last group from 95s onwards is the most difficult and contains all the instances for which an object with a great number of vertices (*helmet*, 1399 vertices) must be extracted.

Figure 4.4: Instances ordered by average execution time of the matheuristic approach.



4.4 Non-convex solid material

A generalization of the EMOC is the problem of **E**xtracting the **M**aximal **O**bject from a **G**eneral Material (EMOG) given that a general material can be either convex or non-convex. The methods presented in Section 4.3 only hold for when the material from which an object is being cut is convex. Indeed, employing the methods from Section 4.3 may lead to infeasible placements such as that illustrated in Figure 4.6(a). Therefore, we develop an adaptation of the EMOC methods to solve the EMOG. When doing so, we make use of the convex hull of the materials, rather than the object itself, as illustrated by the convex hull of a *star*-shaped material in Figure 4.5. Note that the convex hull of an object is the smallest convex set that contains the object.

We propose two methods to solve the problem. The first is an adaptation based on the addition of cutting planes for the matheuristic approach (Section 4.3.2). The second is a MILP model based on Delaunay tetrahedral mesh decomposition

and attempts to optimally solve the EMOG for a fixed orientation (EMOG-FO). For the matheuristic, we consider two variations - with and without local search - where the method without local search implies solving the EMOG-FO.

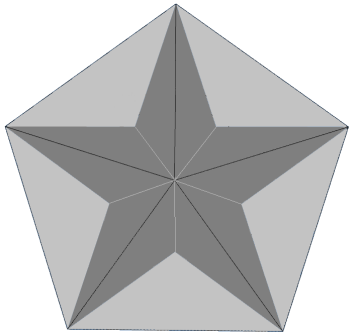


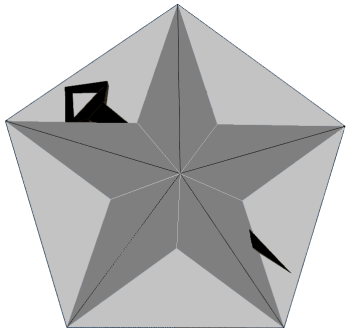
Figure 4.5: Convex hull of a *star*-shaped material.

4.4.1 Matheuristic adaptation for the EMOG

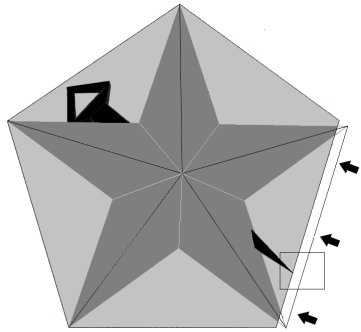
The adapted matheuristic, which will be henceforth referred to as the CP-Matheuristic, follows the same idea as the one used to solve the EMOC. The only difference involves the solution of $LP(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$. Everytime this model is created, it considers the convex hull of the irregular material instead of the material itself. Figure 4.6 provides a visual example of this process. Once a solution is obtained, its feasibility is confirmed by checking if every vertex and plane of the object is inside the irregular material. If a violation is detected, such as in Figure 4.6(a), a cutting plane is added, reducing the original convex hull to an updated convex material in order to push the violating vertices and planes inside the solid material. An example of updating the convex hull in this manner is provided by way of Figures 4.6(b) and 4.6(c). This correction step is repeated at each iteration and the cutting plane procedure is terminated once no violations remain, that is, the object is completely contained within the non-convex material.

There are several strategies for choosing which of the violations to consider at each iteration and which cutting planes to add. Including all cutting planes for all violations at each iteration would result in a much smaller and more innacurate updated convex material, whereas including one plane at a time guarantees more precision, thereby producing final solutions of higher quality. In the case of adding one cutting plane per iteration, several different strategies are still possible. Some examples include:

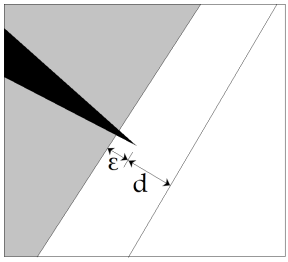
Figure 4.6: Updated convex hull by the addition of cutting planes.



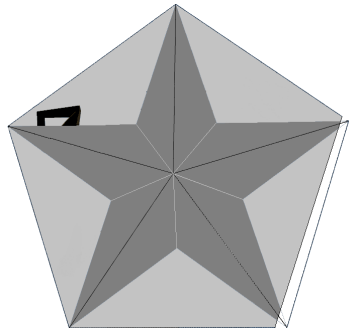
(a) Original convex hull.



(b) Updated convex hull.



(c) Zooming in the infeasible region.



(d) New solution found.

- choose a random plane to move in the convex hull;
- choose the violated plane which is furthest (longest distance) from the convex hull;
- choose the violated plane which is closest (shortest distance) to the convex hull.

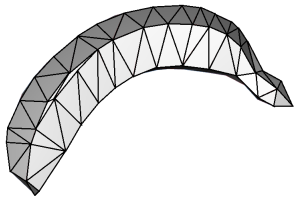
After comparing different strategies for adding cuts, including those listed, we concluded that the strategy achieving the best results for this problem is the following: for each vertex or edge violating the solution we calculate which is closest to the convex hull (or updated convex material boundary) and select this as the candidate vertex or edge. Then, a cutting plane C' is added to the LP (generating $LP'(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$) which pushes the vertices that form the plane which

was closest to the selected vertex/edge to be further inside (Figures 4.6(b) and 4.6(c)), thereby generating an updated convex material which will in its turn produce a new solution (Figure 4.6(d)) which is also checked for new possible infeasibilities. This cutting plane pushes the previous boundary by a distance which corresponds to the length which the point/edge was violating (d) plus a very small constant ($\epsilon = 0.001$, see Figure 4.6(c)). That way, this solution remains outside the feasible region of $LP'(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma})$ and, therefore, will not be generated again. The thought process behind this strategy is that by including cutting planes located close to the violations, the updated convex material remains as similar to its original form as possible, limiting the loss in solution quality.

4.4.2 MILP formulation

The proposed MILP model considers fixed orientations and requires the solid material to be decomposed into convex regions. For this decomposition, we employed the technique proposed by Si (2015). It divides the material using Delaunay decomposition, which generates high quality tetrahedral meshes which together form the original material (Figure 4.7). The source code of Si (2015) is freely available online at `http://www.tetgen.org`.

Figure 4.7: Tetrahedra composing a *banana*-shaped material.



The idea of this formulation is to guarantee that each vertex of the object is located inside one of these tetrahedra. Depending on the shape of the materials or objects, considering only if all vertices of the object are inside the tetrahedra does not guarantee that the entire object is located inside the material. Therefore, everytime this model is solved, a feasibility check must be performed. If, for a given solution, there is an edge of the object located outside the material, this edge is divided in half, introducing an additional new vertex at this midpoint which is also required to be inside one of the tetrahedra of the material. The MILP is then re-solved with these added constraints. This feasibility check is repeated until the solution generated by the model does not contain any infeasibilities, thereby producing the optimal solution for that orientation.

In addition to those from Section 4.3, this model requires some extra parameters and variables to be defined:

Table 4.5: Parameters

$\mathcal{R}$	set of convex regions (tetrahedra) that form the solid material
$\mathcal{F}$	set of all faces from $\mathcal{R}$
$\mathcal{F}_r$	set of faces that form convex region $r \in \mathcal{R}$
n_{xf}, n_{yf}, n_{zf}	coordinates of the normal vector of face $f \in \mathcal{F}$
n_{df}	equals to $(n_{xf}, n_{yf}, n_{zf}) \cdot (x_o, y_o, z_o)$, where (x_o, y_o, z_o) is a point on face $f \in \mathcal{F}$

Table 4.6: Additional decision variables

α_{ir}	binary variable which is equal to 1 if vertex $i \in \mathcal{I}$ is inside convex region $r \in \mathcal{R}$ and 0 otherwise

The problem may then be formulated as follows:

$$
\begin{aligned}
MILP(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}) \quad & \max \ (4.1) \\
& \text{s.t.} \ n_{xf} t_x + n_{yf} t_y + n_{zf} t_z + \\
& \qquad sv_{xi}\bar{\alpha} + sv_{yi}\bar{\beta} + sv_{zi}\bar{\gamma} + && \forall\, i \in \mathcal{I}, \\
& \qquad n_{df} \leq (1 - \alpha_{ir})M, && \forall f \in \mathcal{F}_r, \forall r \in \mathcal{R}, && (4.7) \\
& \sum_{r \in \mathcal{R}} \alpha_{ir} = 1, && \forall\, i \in \mathcal{I}, && (4.8) \\
& n_{xc} t_x + n_{yc} t_y + n_{zc} t_z + \\
& \qquad sv_{xi}\bar{\alpha} + sv_{yi}\bar{\beta} + sv_{zi}\bar{\gamma} + n_{dc} \leq 0, && \forall\, i \in \mathcal{I}, \forall f \in \mathcal{C}, && (4.9) \\
& \alpha_{ir} \in \{0, 1\}, && \forall\, i \in \mathcal{I}, && (4.10) \\
& (4.3), (4.4).
\end{aligned}
$$

Constraints (4.7) check, for each convex region r of the material, if vertex i is inside r, while Constraints (4.8) ensure that each vertex i is inside exactly one convex region. Constraints (4.9) are included as valid inequalities to improve the upper bounds of the model. They ensure that all vertices of the object are located inside the convex hull of the solid material.

Considering that the size of this model depends on the number of tetrahedra generated and additional new vertices included, problems with complex shapes become very difficult to solve within an acceptable runtime.

4.4.3 Computational study

Comparison of MILP against the CP-Matheuristic for fixed orientations

When the material is non-convex, we evaluate the quality of the CP-Matheuristic by comparing its results against those of the MILP. Similar to the NLP models of the EMOC, complex outer objects (corresponding to too many tetrahedra) generate MILP models with too many variables and constraints and, therefore, only small instances could be solved. In this case we choose three specific shapes as solid materials - *apple*, *banana* and *star* - which are decomposed into 244, 283 and 13 tetrahedra, respectively, the three smallest numbers of tetrahedra among the instances.

In order to compare the performance of the MILP and the CP-Matheuristic for fixed rotations, we run the two methods with ten randomly generated rotations. Note that for the matheuristic this corresponds to solving the LP together with the cutting planes for a fixed rotation. The computer hardware is the same as the previous experiments and we limit the total runtime permitted to solve the MILP to 3600s. It is important to recall that solutions produced by the MILP within the time limit are optimal for the chosen orientation. Table 4.7 details the average performance of each method over the ten executions.

Table 4.7: MILP model compared to matheuristic approach for the EMOG-FO.

Items		*MILP*		**CP-Matheuristic**		
Material	Object	avg vol	avg t(s)	avg vol	avg t(s)	avg gap(%)
apple	dagger	**445.50**	1710.1	351.45	0.3	21.11
apple	star	**1236.69**	45.0	1064.13	0.1	13.99
apple	tree	**7282.17**	1890.6	7280.05	0.1	0.03
banana	dagger	0.01	3600.0	**0.31**	0.1	-3000.0
banana	star	**0.14**	959.4	0.10	0.1	28.57
banana	tree	0.57	3600.0	**0.70**	0.1	-22.81
star	dagger	**186.54**	23.3	128.99	0.1	30.85
star	apple	**1842.14**	2.6	1631.92	0.1	11.41
star	tree	**480.82**	21.4	469.25	0.1	2.41

Many interesting conclusions can be drawn concerning the performance of both the matheuristic and the formulation. Beginning with the matheuristic, one weakness of the proposed strategy is that for heavily-indented non-convex materials, heuristically adding cutting planes can dramatically impair solution

quality, as is evident when trying to extract maximal objects from the *star*, particularly the maximal *dagger*. On the other hand, as already mentioned, the quality of the solutions produced by the MILP are limited due to the size of the model being solved. Considering the *banana* as the solid material, which has a greater number of tetrahedra in its decomposition compared to the *star* and the *apple*, the model struggles, reaching the time limit for most of the executions and producing very poor results.

For the case with the *apple* as the material shape which, in terms of model size, lies between the *dagger* and the *banana*, a clear increase in execution time can be observed compared to when the *star* is employed as the material shape (the smallest instance). It is important to mention that although the average solution values for these instances were better for the MILP model, in approximately one-third of the executions the CP-Matheuristic outperformed the MILP.

Computational study on larger instances

In this section, we solve more complicated instances using the CP-Matheuristic. The materials consist of the non-convex shapes representing stones. In order to generate random irregular 3D shapes, we use the alpha shape functions of MATLAB. This function requires a set of points and an alpha radius value, say R, for creating the alpha shapes. Therefore, we begin by generating a set of random points in 3D. Let N be the number of points that we want to generate and X, Y and Z refer to the coordinates of the Euclidean space. We generate N random numbers by using a different probability distribution for each coordinate. More explicitly, let $\{x_1, x_2, \ldots, x_N\}, \{y_1, y_2, \ldots, y_N\}$, and $\{z_1, z_2, \ldots, z_N\}$ be the numbers we generate for X, Y and Z coordinates. Then, $P = \{(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_N, y_N, z_N)\}$ is the set of 3D points we obtain.

In order to obtain a variety of alpha shapes, we use different combinations of parameter values which are not necessarily identical for X, Y and Z. We apply this procedure to produce a set of stone-shaped materials from which seven were selected.

The objects selected are also seven: *star*, *tree*, *dragon*, *dagger*, *pig*, *apple* and *banana*. The *dragon* and the *pig* objects were taken from Lien (2009). The idea of these experiments is to evaluate the matheuristic in its complete form, where the orientation of the object is gradually changed at every step of the local search method.

As in the previous experiments, Gurobi 8.1 was used to solve the LP model in combination with the cutting planes. Table 4.8 details the average volumes of

the maximal objects obtained for each instance, together with their respective standard deviation after ten runs, the average percentage of volume occupied in the solid materials and the average execution time of the algorithm.

Table 4.8: Matheuristic performance for the EMOG with instances containing non-convex solid materials.

Items		**Matheuristic**				**Items**		**Matheuristic**			
Material	Object	vol	std dev	occ(%)	t(s)	Material	Object	vol	std dev	occ(%)	t(s)
stone 1	star	79.24	0.00	3.3	7.6	stone 5	star	79.24	0.00	3.2	6.1
stone 1	tree	293.59	0.00	12.0	31.7	stone 5	tree	321.10	0.00	13.1	10.3
stone 1	dragon	235.08	5.93	9.6	128.3	stone 5	dragon	242.50	4.47	9.9	87.4
stone 1	dagger	65.01	0.00	2.7	16.1	stone 5	dagger	89.18	0.00	3.6	15.2
stone 1	pig	477.28	5.08	19.6	117.8	stone 5	pig	462.15	9.90	18.8	105.1
stone 1	apple	675.32	0.00	27.7	102.0	stone 5	apple	767.29	0.00	31.2	56.62
stone 1	banana	414.36	0.00	17.0	16.9	stone 5	banana	445.29	5.26	18.1	9.3
stone 2	star	0.23	0.00	2.5	6.6	stone 6	star	0.78	0.00	3.12	6.8
stone 2	tree	1.76	0.00	19.2	12.4	stone 6	tree	2.80	0.00	11.2	11.9
stone 2	dragon	1.01	0.04	11.0	81.9	stone 6	dragon	2.53	0.07	10.1	85.08
stone 2	dagger	0.09	0.00	1.0	16.7	stone 6	dagger	0.71	0.00	2.8	16.3
stone 2	pig	1.38	0.00	15.0	106.4	stone 6	pig	5.13	0.25	20.5	109.8
stone 2	apple	3.55	0.00	38.7	21.8	stone 6	apple	6.94	0.00	27.8	24.4
stone 2	banana	1.08	0.00	11.8	9.4	stone 6	banana	4.57	0.24	18.3	9.5
stone 3	star	9.90	0.00	4.2	7.1	stone 7	star	0.78	0.00	2.1	5.5
stone 3	tree	14.12	0.00	6.0	16.9	stone 7	tree	5.96	0.00	16.2	9.7
stone 3	dragon	13.03	0.21	5.5	118.3	stone 7	dragon	3.96	0.10	10.8	73.4
stone 3	dagger	8.13	0.00	3.4	18.2	stone 7	dagger	0.30	0.00	0.8	14.3
stone 3	pig	48.36	1.11	20.4	117.3	stone 7	pig	3.78	0.00	10.3	106.6
stone 3	apple	40.46	0.00	17.1	37.82	stone 7	apple	19.04	0.00	51.7	9.8
stone 3	banana	49.93	1.22	21.0	10.7	stone 7	banana	3.23	0.00	1.2	8.1
stone 4	star	9.91	0.00	3.2	6.6						
stone 4	tree	33.46	0.00	10.8	10.5						
stone 4	dragon	29.02	0.36	9.3	80.9						
stone 4	dagger	8.13	0.00	2.6	16.2						
stone 4	pig	60.30	0.00	19.4	104.3						
stone 4	apple	95.91	0.00	30.8	30.8						
stone 4	banana	55.00	1.32	17.7	8.8						

The results confirm the effectiveness of the proposed method, which is capable of solving instances with a large variety of shape combinations within a short average runtime (maximum average time is 128.3s). The variety of shapes is shown by the variation of volume occupations in the materials. It becomes clear that the most time-consuming instances are those which involve either the *dragon* or *pig*. Note that these are the objects with the greatest number of vertices, confirming the discussion about execution time being affected by the size of objects considered (Section 4.3). In terms of robustness of our proposed matheuristic, the majority of the results had a standard deviation of 0.00 and the worst value is for instance *stone 1* x *dragon* (5.93), which is also the most demanding in terms of average time, verifying its difficulty. The main purpose of these experiments is to establish benchmark instances for future research and therefore all the objects used in this chapter have been made publicly available at `http://dx.doi.org/10.17632/phhzsbpkj9.1`.

4.5 Conclusion

This chapter proposed a well-performing matheuristic approach to both the problem of extracting the maximal object from a **convex** solid material and its generalization, the problem of extracting the maximal object from a **general** material. The experiments demonstrated the efficiency of the approach, with the results outperforming all compared approaches both in terms of solution quality and time. We also confirmed the robustness of the method, generating identical solution values accross ten different runs for all instances except one. Not only this does constitute an important contribution, applicable to any related problem (Section 4.2), it also builds up as a base for future improvements such as more precise representations of the feasible region of non-convex shapes.

In order to address a wide range of applications, all instances were created consisting of objects and materials of various characteristics. These instances are provided online so that they can be used as benchmark for future research. For future work, one could extend this approach for the case where multiple objects must be extracted from the same solid material.

Chapter 5

Conclusion

Although generally well-solved by human decision makers, realistic three-dimensional cutting and packing problems still require an extended period of practice and training for those individuals to generate good quality solutions. Moreover, despite their expertise, some difficult cases may take weeks for high quality solutions to be generated. Considering the limited and problem-specific literature, this thesis addressed how to steer 3D cutting and packing research away from unrealistic academic models towards accommodating real-world conditions and limitations. By combining the advantages of approaches from the literature, we were able to develop robust solution methods capable of solving multiple variants of these problems, from regular boxes and containers to more extreme irregular cases.

Chapter 2 engaged in a different way of performing a literature review by adapting, re-implementing and comparing on up-to-date hardware the most significant exact methods in the literature of 3D regular packing problems. Besides this already important contribution, given the lack of such a review, we were expressly interested in determining what exactly those methods had in common rather than which necessarily performed best. When adapting each method to solve the same problem variant it became clear that it is straightforward to modify those mathematical formulations in order to address different variants. As expected, the results demonstrated that purely mathematical formulations are still some time away from being effectively employed to solve real-world problems, but they can definitely already be employed as a component of a solution method.

Several other important conclusions were drawn from this chapter which will inevitably lead to some new interesting lines of research with respect to exact

solution methods. For example, considering approaches that focus on an indirect representation of the relative positions between the boxes instead of the direct approach of each box's coordinates is something which has not been accomplished as of yet. Another point that should be further investigated is the development of new formulations focused on specific problem variants. Taking into account problem specifics tends to provide important advantages when developing a solution method. This review also showed how important it is to properly compare solution methods, showing how it is not necessarily always the case that newly proposed methods are the best performing and that a fair comparison is always required in order to correctly evaluate which performs the best in which situations.

After analyzing and experimenting different solution methods, Chapter 3 provided a matheuristic framework for solving the 3D single container problem when considering weakly heterogeneous boxes. The matheuristic was able to solve problems reflective of those found in the logistics industry by considering constraints that cover various practical aspects of the problem: container-related (weight limit, load balance), item-related (orientation, loading priorities, stacking), positioning- or load-related (stability) constraints. We were able to demonstrate how to solve a problem considering a number of real-world constraints, something previously missing for this problem variant. Moreover, we were able to demonstrate how flexible the solution approach is by generating good quality results when experimenting with different subsets of real-world constraints and adapting the matheuristic to efficiently solve an extension of the problem which considers multiple identical containers.

Some variants do, however, require more substantial modifications. To address the problem variants that deal with strongly heterogeneous boxes, modifying the grouping structure for boxes is mandatory. The wall structure used in our approach is only effective for variants involving identical or weakly heterogeneous boxes. Therefore, although we cannot guarantee that the developed approach will efficiently solve every single 3D packing problem variant when applied exactly the way it is reported, we provide a powerful new research angle from which to address and solve such problems. Furthermore, research concerning which grouping structures to adopt in each specific problem and what changes would be necessary in the remaining components of the solution approach are themselves very interesting starting points for future research.

We hope that the research developed and the conclusions taken from Chapters 2 and 3 establish some solid foundations for further research concerning realistic regular 3D packing problems. Considering more extreme irregular cases, Chapter 4 addressed the problem of extracting maximal objects from solid materials. This problem relates to a wide set of applications such as the manufacture of integrated circuits, diamond, gemstone cutting and irregular object packing

(3D printing, for example) and each application has developed its own specific literature. Therefore, we aimed at solving a more general problem in order to provide a useful base for all applications. It was verified that a similar strategy as the proposed matheuristic is indeed possible to address a related and relevant problem with several practical applications. The instances developed to test and ultimately accomplish this goal have been made available online and provide a set of benchmark instances for future research.

Finally, this thesis accomplished what ought to be expected from academic research: successfully addressing current research gaps and providing forward guidance for future research. Our solution approaches bridged the gap between academia and practice with respect to 3D cutting and packing problems. We genuinely hope that industrial and academic decision makers will take the research reported in this thesis into account when addressing challenging real-world 3D cutting and packing problems.

Appendix A

SSSCSP formulations

Knapsack and Packing problems

Knapsack formulation for the SSSCSP problem (Chapter 3):

$$\min \quad \sum_{j \in \mathcal{J}} c_j \cdot u_j \tag{A.1}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} l_i \cdot p_{ij} \leq L, \tag{A.2}$$

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} d_{ti} \cdot p_{ij} = \mathcal{D}_t, \qquad \forall\, t \in \mathcal{T}, \tag{A.3}$$

$$\sum_{i \in \mathcal{I}} p_{ij} \leq M \cdot u_j, \qquad \forall\, j \in \mathcal{J}, \tag{A.4}$$

$$u_j \geq u_{j+1}, \qquad \forall\, j \in \mathcal{J}. \tag{A.5}$$

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} m_i \cdot p_{ij} \leq P_{max}, \tag{A.6}$$

$$u_j \in \{0, 1\}, \qquad \forall\, j \in \mathcal{J}, \tag{A.7}$$

$$p_{ij} \in \mathbb{Z}_+, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J}. \tag{A.8}$$

Packing formulation for the SSSCSP problem:

$$\min \sum_{j \in \mathcal{J}} c_j \cdot u_j \tag{A.9}$$

$$\text{s.t. } x_i + l_i \leq x_k + (1 - a_{ik}) \cdot L, \qquad \forall\, i,k \in \mathcal{I}, i < k, \tag{A.10}$$

$$x_k + l_k \leq x_i + (1 - b_{ik}) \cdot L, \qquad \forall\, i,k \in \mathcal{I}, i < k, \tag{A.11}$$

$$a_{ik} + b_{ik} \geq s_{ij} + s_{kj} - 1, \qquad \forall\, i,k \in \mathcal{I}, i < k,$$

$$\forall\, j \in \mathcal{J}, \tag{A.12}$$

$$x_i + l_i \leq L + (1 - s_{ji}) \cdot L, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J}, \tag{A.13}$$

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} d_{ti} \cdot s_{ij} = \mathcal{D}_t, \qquad \forall\, t \in \mathcal{T}, \tag{A.14}$$

$$\sum_{i \in \mathcal{I}} s_{ij} \leq M \cdot u_j, \qquad \forall\, j \in \mathcal{J}, \tag{A.15}$$

$$u_j \geq u_{j+1}, \qquad \forall\, j \in \mathcal{J}. \tag{A.16}$$

$$\sum_{i \in \mathcal{I}} m_i \cdot s_{ij} = P_{cargo_j}, \qquad \forall\, j \in \mathcal{J}, \tag{A.17}$$

$$P_{cargo_j} \leq P_{max}, \qquad \forall\, j \in \mathcal{J}, \tag{A.18}$$

$a_{ik} = 0$, if wall i contains box type t_1, wall k contains

boxt type t_2 and a positioning constraint between $\qquad \forall\, t_1, t_2 \in \mathcal{T}, t_1 \neq t_2,$

box types t_1 and t_2 is active, $\qquad \forall\, i,k \in \mathcal{I}, i < k,$ (A.19)

$b_{ik} = 0$, if wall i contains box type t_2, wall j contains

box type t_1 and a positioning constraint between $\qquad \forall\, t_1, t_2 \in \mathcal{T}, t_1 \neq t_2,$

box types t_1 and t_2 is active, $\qquad \forall\, i,k \in \mathcal{I}, i < k,$ (A.20)

$$g_{ij} \leq L \cdot s_{ij}, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J}, \tag{A.21}$$

$$g_{ij} \leq x_{ij} + \frac{l_i}{2}, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J}, \tag{A.22}$$

$$g_{ij} \geq x_{ij} + \frac{l_i}{2} + L \cdot s_{ij} - L, \qquad \forall\, j \in \mathcal{J}, \tag{A.23}$$

$$\sum_{i \in \mathcal{I}} m_i \cdot g_{ij} = NC_{xj}, \qquad \forall\, j \in \mathcal{J}, \tag{A.24}$$

$$NC_{x_j} \leq P_{cargo_j} \cdot (TWB - H_d) - TWB \cdot FT_{max} \qquad \forall\, j \in \mathcal{J},$$

$$+ U \cdot (TWB - H_u), \qquad \forall\, j \in \mathcal{J}, \tag{A.25}$$

$$NC_{x_j} \leq TWB \cdot RT_{max} - U \cdot H_u - P_{cargo} * H_d, \qquad \forall\, j \in \mathcal{J}, \tag{A.26}$$

$$NC_{x_j} \geq P_{cargo} \cdot (H_d - S_{min} \cdot TWB - TWB) \qquad \forall\, j \in \mathcal{J},$$

$$+ U \cdot (TWB - H_u - S_{min} \cdot TWB), \qquad \forall\, j \in \mathcal{J}, \tag{A.27}$$

$$NC_{x_j} \geq U \cdot (T_{min} \cdot TWB - H_u) \qquad \forall\, j \in \mathcal{J},$$

$$+ P_{cargo} \cdot (T_{min} \cdot TWB - H_d), \qquad \forall\, j \in \mathcal{J}, \tag{A.28}$$

$$u_j, s_{ij} \in \{0, 1\}, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J} \tag{A.29}$$

$$a_{ik}, b_{ik} \in \{0, 1\}, \qquad \forall\, i, k \in \mathcal{I}, \tag{A.30}$$

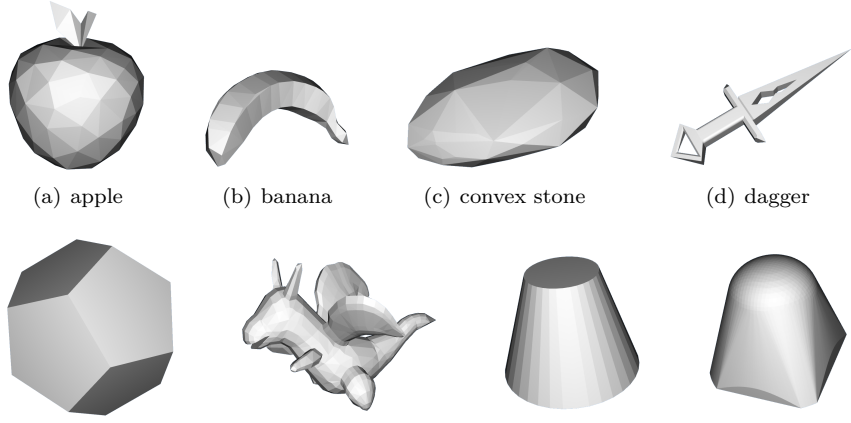
$$x_{ij}, g_{ij}, P_{cargo_j}, NC_{x_j}, \geq 0, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J}. \tag{A.31}$$

Appendix B

EMOC/EMOG instances

Objects details

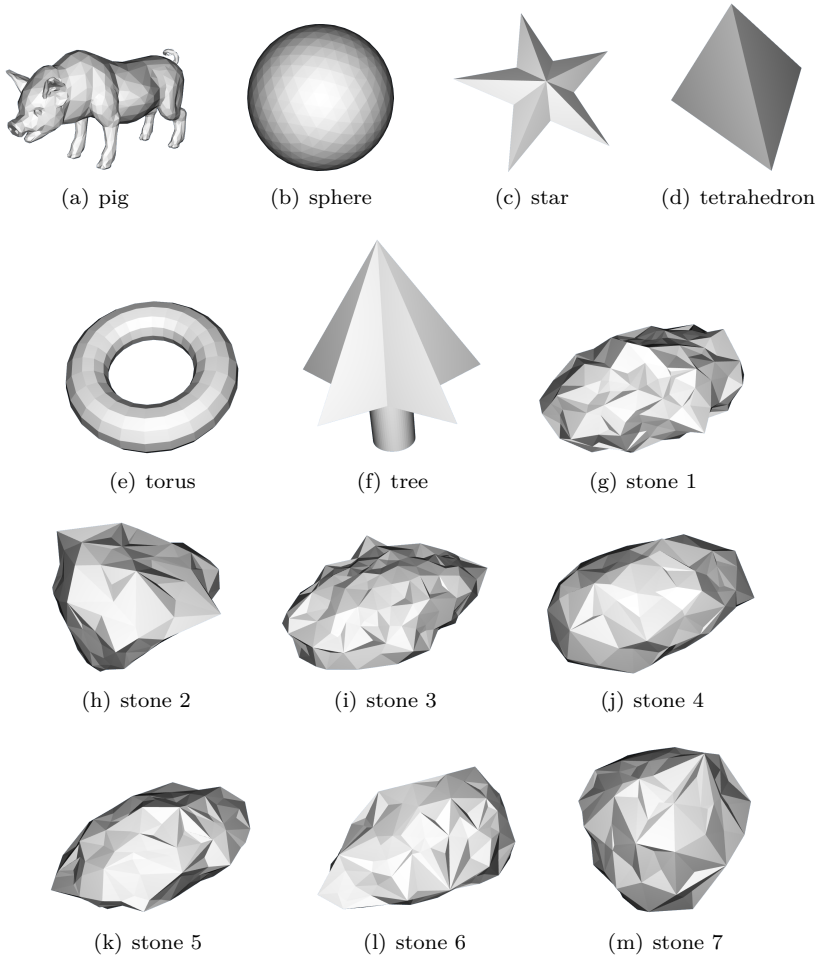
Materials and objects used for the experiments in Chapter 4 are displayed in Figures B.1 and B.2 while the respective number of vertices and faces of each object are presented in Table B.1.



(a) apple (b) banana (c) convex stone (d) dagger

(e) dodecahedron (f) dragon (g) frustum (h) helmet

Figure B.1: Solid materials and objects (part 1).



(a) pig (b) sphere (c) star (d) tetrahedron

(e) torus (f) tree (g) stone 1

(h) stone 2 (i) stone 3 (j) stone 4

(k) stone 5 (l) stone 6 (m) stone 7

Figure B.2: Solid materials and objects (part 2).

Items	**Vertices**	**Faces**
apple	103	200
banana	102	200
convex stone	70	136
dagger	368	740
dodecahedron	32	60
dragon	1166	2328
frustum	74	144
helmet	1399	2794
pig	1394	2784
sphere	642	1280
star	16	28
stone 1	306	608
stone 2	278	552
stone 3	374	744
stone 4	219	434
stone 5	225	446
stone 6	295	586
stone 7	167	330
tetrahedron	4	4
torus	288	576
tree	156	308

Table B.1: Number of vertices and faces of each item.

Bibliography

Abdel-Malek, H. L. and A. S. O. Hassan (1991). "The ellipsoidal technique for design centering and region approximation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 10.8, pp. 1006–1014.

Allen, S. D., E. K. Burke, and J. Mareček (2012). "A space-indexed formulation of packing boxes into a larger box". In: *Operations Research Letters* 40, pp. 20–24.

Alvarez-Valdes, R., M. A. Carravilla, and J. F. Oliveira (2018). "Cutting and Packing". In: *Handbook of Heuristics*. Ed. by R. Martí, P. Panos, and M. Resende. Springer Cham, pp. 1–46.

Araya, I., K. Guerrero, and E. Nuñez (2017). "VCS: A new heuristic function for selecting boxes in the single container loading problem". In: *Computers & Operations Research* 82, pp. 27–35.

Araya, I. and M. C. Riff (2014). "A beam search approach to the container loading problem". In: *Computers & Operations Research* 80, pp. 100–107.

Bennell, J. A. and J. F. Oliveira (2008). "The geometry of nesting problems: a tutorial". In: *European Journal of Operational Research* 184, pp. 397–415.

Bischoff, E. E. (2006). "Three-dimensional packing of items with limited load bearing strength". In: *European Journal of Operational Research*, pp. 952–966.

Bischoff, E. E. and M. S. W. Ratcliff (1995). "Issues in the development of approaches to container loading". In: *European Journal of Operational Research* 80, pp. 68–76.

Bortfeldt, A. and G. Wäscher (2013). "Constraints in container loading - A state-of-the-art review". In: *European Journal of Operational Research* 229, pp. 1–20.

Bortfeldt, A. and H. Gehring (1999). "Zur Behandlung von Restriktionen bei der Stauraumoptimierung am Beispiel eines genetischen Algorithmus für das Containerbeladeproblem". In: *Logistik Management: Intelligente I + K Technologien*. Ed. by H. Kopfer and C. Bierwirth. Springer Berlin Heidelberg, pp. 83–100.

Chen, C. S., S. M. Lee, and Q. S. Shen (1995). “An analytical model for the container loading problem”. In: *European Journal of Operational Research* 80.1, pp. 68–76.

Chernov, N., Y. Stoyan, and T. Romanova (2010). “Mathematical model and efficient algorithms for object packing problem”. In: *Computational Geometry* 43.5, pp. 535–553.

Cherri, A. C., M. N. Arenales, H. H. Yanasse, K. C. Poldi, and A. C. G. Vianna (2014). “The one-dimensional cutting stock problem with usable leftovers - A survey”. In: *European Journal of Operational Research* 236.2, pp. 395–402.

Christensen, S. G. and Rousøe (2009). “Container loading with multi-drop constraints”. In: *International Transactions in Operational Research* 16, pp. 727–743.

Crainic, T. G., G. Perboli, and R. Tadei (2009). “TS2PACK: a two-level tabu search for the three-dimensional bin packing problem”. In: *European Journal of Operational Research* 195.3, pp. 774–760.

Da Silva, V. B., M. Ritt, J. B. d. P. Carvalho, M. J. Brusso, J. T. da Silva, and A. L. Zanatta (2012). “A real-valued genetic algorithm for gemstone cutting”. In: *2012 XXXVIII Conferencia Latinoamericana En Informatica (CLEI)*. Centro Latinoamericano de Estudios en Informática, pp. 1–8.

den Boef, E., J. Korst, S. Martello, D. Pisinger, and D. Vigo (2005). “Erratum to "Three-Dimensional Bin Packing Problem": Robot-Packable and Orthogonal Variants of Packing Problems”. In: *Optimization Letters* 53.4, pp. 735–736.

Director, S. W. and G. D. Hachtel (1977). “The simplicial approximation approach to design centering”. In: *IEEE Transactions on Circuits and Systems* 24.7, pp. 363–372.

Egeblad, J., C. Garavelli, S. Lisi, and D. Pisinger (2010). “Heuristics for container loading of furniture”. In: *European Journal of Operational Research* 200, pp. 881–892.

Egeblad, J., B. K. Nielsen, and A. Odgaard (2007). “Fast neighborhood search for two-and three-dimensional nesting problems”. In: *European Journal of Operational Research* 183.3, pp. 1249–1266.

Eley, M. (2002). “Solving container loading problems by block arrangement”. In: *European Journal of Operational Research* 141, pp. 393–409.

Elhedhli, S., F. Gzara, and Y. F. Yan (2017). “A MIP-based slicing heuristic for three-dimensional bin packing”. In: *Optimization Letters*, pp. 1–17.

Euler, L. (1776). “Formulae generales pro translatione quacunque corporum rigidorum”. In: *Novi Commentarii academiae scientiarum petropolitanae* 20, pp. 189–207.

Fanslau, T. and A. Bortfeldt (2010). “A tree search algorithm for solving the Container Loading Problem”. In: *INFORMS Journal on Computing* 22.2, pp. 222–235.

Fasano, G. (2004). “A MIP approach for some practical packing problems: balancing constraints and tetris-like items”. In: *Quarterly Journal of the Belgian, French and Italian Operations Research Societies* 2.2, pp. 161–174.

Gonçalves, J. F. and M. G. C. Resende (2012). “A parallel multi-population biased random-key genetic algorithm for a container loading problem”. In: *Computers & Operations Research* 39, pp. 179–190.

Hassan, A. S. O. and A. Abdel-Naby (2012). “A new hybrid method for optimal circuit design using semi-definite programming”. In: *Engineering Optimization* 44.6, pp. 725–740.

He, K. and W. Huang (2011). “An efficient placement heuristic for three-dimensional rectangular packing”. In: *Computers & Operations Research* 38, pp. 227–233.

Hifi, M. (2002). “Approximate algorithms for the container loading problems”. In: *International Transactions in Operational Research* 9, pp. 747–774.

Hifi, M., I. Kacem, S. Nègre, and L. Wu (2010). “A linear programming approach for the three-dimensional bin-packing problem”. In: *Electronic Notes in Discrete Mathematics* 36, pp. 993–1000.

Ivancic, N., K. Mathur, and B. B. Mohanty (1989). “An integer programming based heuristic approach to the three-dimensional packing problem”. In: *Journal of Manufacturing and Operations Management* 2, pp. 268–298.

Junqueira, L., R. Morabito, and D. S. Yamashita (2012). “Three-dimensional container loading models with cargo stability and load bearing constraints”. In: *Computers & Operations Research* 39, pp. 74–85.

Küfer, K.-H., V. Maag, and J. Schwientek (2015). “Maximal Material Yield in Gemstone Cutting”. In: *Currents in Industrial Mathematics*. Springer, pp. 229–289.

Layeb, S. B., O. Jabloun, and A. Jaoua (2017). “New heuristic for the Single Container Loading Problem”. In: *International Journal of Economics & Strategic Management of Business Process* 8, pp. 1–7.

Leao, A. A. S., F. M. B. Toledo, J. F. Oliveira, M. A. Carravilla, and R. Alvarez-Valdés (2019). “Irregular packing problems: a review of mathematical models”. In: *European Journal of Operational Research*.

Lien, J.-M. (2009). “A simple method for computing Minkowski sum boundary in 3D using collision detection”. In: *Algorithmic Foundation of Robotics VIII*. Ed. by G. S. Chirikjian, H. Choset, M. Morales, and T. Murphey. Springer Tracts in Advanced Robotics, pp. 401–415.

Lim, A., H. Ma, J. Xu, and X. Zhang (2012). “An iterated construction approach with dynamic priorization for solving the container loading problems”. In: *Expert Systems with Applications* 39, pp. 4292–4305.

Liu, J., Y. Yue, Z. Dong, C. Maple, and M. Keech (2011). “A novel hybrid tabu search to container loading”. In: *Computers & Operations Research* 38, pp. 797–807.

Loh, T. and A. Nee (1992). “A packing algorithm for hexandral boxes”. In: *Proceedings of the Conference of Industrial Automation* 2, pp. 115–126.

Lorie, J. H. and L. J. Savage (1955). “Three problems in capital rationing”. In: *The Journal of Business* 28, pp. 229–239.

Lourenço, H. R. and O. C. Martin (2003). “Iterated Local Search”. In: *Handbook of Metaheuristics*. Ed. by F. Glover and G. A. Kochenberger. Springer, pp. 320–353.

Ma, Y., Z. Chen, W. Hu, and W. Wang (2018). “Packing irregular objects in 3D space via hybrid optimization”. In: *Computer Graphics Forum*. Vol. 37. Wiley Online Library, pp. 49–59.

Maag, V. (2015). “A collision detection approach for maximizing the material utilization”. In: *Computational Optimization and Applications* 61.3, pp. 761–781.

Markowitz, H. M. and A. S. Manne (1957). “On the solution of discrete programming problems”. In: *Econometrica* 25, pp. 84–110.

Martello, S., D. Pisinger, and D. Vigo (2000). “The three-dimensional bin packing problem”. In: *Operations Research* 48.2, pp. 256–267.

Martello, S., D. Pisinger, D. Vigo, E. den Boef, and J. Korst (2007). “Algorithm 864: General and Robot-Packable Variants of the Three-Dimensional Bin Packing Problem”. In: *ACM Transactions on Mathematical Software* 33.1, pp. 1–12.

Mohanty, B. B., K. Mathur, and N. J. Ivancic (1994). “Value considerations in three-dimensional packing - a heuristic procedure using the fractional knapsack problem”. In: *European Journal of Operational Research* 74.1, pp. 143–151.

Moon, I. and T. V. L. Nguyen (2014). “Container packing problem with balance constraints”. In: *OR Spectrum* 36, pp. 837–878.

Nguyen, V. H. and J.-J. Strodiot (1992). “Computing a global optimal solution to a design centering problem”. In: *Mathematical Programming* 53.1, pp. 111–123.

Olsson, J., T. Larsson, and N.-H. Quttineh (2020). “Automating the planning of container loading for Atlas Copco: Coping with real-life stacking and stability constraints”. In: *European Journal of Operational Research* 280, pp. 1018–1034.

Paquay, C., M. Schyns, and S. Limbourg (2016). “A mixed integer programming formulation for the three-dimensional bin packing problem deriving from an air cargo application”. In: *International Transactions in Operational Research* 23.1-2, pp. 187–213.

Parreño, F., R. Alvarez-Valdes, J. Tamarit, and J. F. Oliveira (2008). “A maximal-space algorithm for the container loading problem”. In: *INFORMS Journal on Computing* 20, pp. 412–422.

Ramos, A. G., E. Silva, and J. F. Oliveira (2018). “A new load balance methodology for container loading problem in road transportation”. In: *European Journal of Operational Research* 266, pp. 1140–1152.

Ren, J., Y. Tian, and T. Sawaragi (2011). "A tree search method for the container loading problem with shipment priority". In: *European Journal of Operational Research* 214, pp. 526–535.

Romanova, T., J. Bennell, Y. Stoyan, and A. Pankratov (2018). "Packing of concave polyhedra with continuous rotations using nonlinear optimisation". In: *European Journal of Operational Research* 268.1, pp. 37–53.

Sapatnekar, S. S., P. M. Vaidya, and S.-M. Kang (1994). "Convexity-based algorithms for design centering". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13.12, pp. 1536–1549.

Saraiva, R. D., N. Nepomuceno, and R. Pinheiro (2019). "A two-phase approach for single container loading with weakly heterogeneous boxes". In: *Algorithms* 12, pp. 67–82.

Si, H. (2015). "TetGen, a Delaunay-based quality tetrahedral mesh generator". In: *ACM Transactions on Mathematical Software* 41.2, 11:1–11:36.

Silva, E. F., T. A. M. Toffolo, and T. Wauters (2019). "Exact methods for three-dimensional cutting and packing: A comparative study". In: *Computers and Operations Research* 109, pp. 12–27.

Silva, E., A. G. Ramos, and J. F. Oliveira (2018). "Load balance recovery for multi-drop distribution problems: A mixed integer linear programming approach". In: *Transportation Research Part B* 116, pp. 62–75.

Silva, E., J. F. Oliveira, and G. Wäscher (2014). "2DCPackGen: a problem generator for two-dimensional rectangular cutting and packing problems". In: *European Journal of Operational Research* 237, pp. 846–856.

Stoyan, Y. G., N. I. Gil, G. Scheithauer, A. Pankratov, and I. Magdalina (2005). "Packing of convex polytopes into a parallelepiped". In: *Optimization* 54.2, pp. 215–235.

Toffolo, T. A. M., E. Esprit, T. Wauters, and G. Vanden Berghe (2017). "A two-dimensional heuristic decomposition approach to a three-dimensional multiple container loading problem". In: *European Journal of Opertional Reserach* 257.2, pp. 526–538.

Trucks, V. (2015). *Data sheet specification model range FE 62TR.* `http://segotn12827.rds.volvo.com/stpifiles/volvo/modelrange/fe62tra_prt_prt.pdf`. Accessed: July 26, 2019.

Tsai, J.-F., P.-C. Wang, and M.-H. Lin (2015). "A global optimization approach for solving three-dimensional open dimension rectangular packing problems". In: *Optimization* 64.12, pp. 2601–2618.

Wäscher, G., H. Haußner, and H. Schumann (2007). "An improved topology of cutting and packing problems". In: *European Journal of Operational Research* 183.3, pp. 1109–1130.

Winterfeld, A. (2008). "Application of general semi-infinite programming to lapidary cutting problems". In: *European Journal of Operational Research* 191.1, pp. 838–854.

Wojciechowski, J. M. and J. Vlach (1993). "Ellipsoidal method for design centering and yield estimation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 12.10, pp. 1570–1579.

Zhao, X., J. A. Bennell, T. Bektaş, and K. Dowsland (2016). "A comparative review of 3D container loading algorithms". In: *International Transactions in Operational Research* 23, pp. 287–320.

Zhu, W., W. Huang, and A. Lim (2012). "A prototype column generation strategy for the multiple container loading problem". In: *European Jouranl of Operational Research* 223, pp. 27–39.

Zhu, W. and A. Lim (2012). "A new iterative-doubling Greedy-Lookahead algorithm for the single container loading problem". In: *European Journal of Operational Research* 222, pp. 408–417.

List of publications

Publications

Articles in internationally reviewed academic journals

2019 **Silva, E. F.**, Toffolo, T. A. M., Wauters, T. Exact methods for three-dimensional cutting and packing: A comparative study concerning single container problems. *Computers & Operations Research*, 109:12-27. doi:10.1016/j.cor.2019.04.020

2019 **Silva, E. F.**, Oliveira, L. T., Oliveira, J. F., Toledo, F. M. B. Exact approaches for the cutting path determination problem. *Computers & Operations Research*, 112:104772. doi:10.1016/j.cor.2019.104772

Conferences

Presentations at conferences and scientific meetings

2019 **Silva, E. F.**, Leão, A. S., Toledo, F. M. B., Wauters, T. A matheuristic for the 3D Single Large Object Placement Problem. *16th EURO Special Interest Group on Cutting and Packing Meeting.* ESICUP Meeting 2019, Mexico City, Mexico - April 10, 2019.

2018 **Silva, E. F.**, Wauters, T. A parallel tree based approach for the 3D Cutting and Packing Problem. *29th European Conference on Operational Research.* EURO 2018, Valencia, Spain - July 11, 2018.

2018 **Silva, E. F.**, Wauters, T. A parallel Branch and Bound for the 3D Cutting and Packing Problem. *15th EURO Special Interest Group on Cutting and Packing Meeting.* ESICUP Meeting 2018, Zoetermeer, The Netherlands - May 25, 2018.

2017 **Silva, E. F.**, Toffolo, T. A. M., Vanden Berghe, G., Wauters, T. Improving exact methods for solving 3D cutting and packing problems. *XII International Workshop on Cutting, Packing and Related Topics.* IWCPRT 2017, Gent, Belgium - September 11, 2017.

2017 **Silva, E. F.**, Toffolo, T. A. M., Vanden Berghe, G., Wauters, T. A comparison between exact methods for 3D packing problems. *14th EURO Special Interest Group on Cutting and Packing Meeting.* ESICUP Meeting 2017, Liège, Belgium - May 04, 2017.

2017 **Silva, E. F.**, Toffolo, T. A. M., Vanden Berghe, G., Wauters, T. A study of exact methods for container loading problems. *31st Annual Conference of the Belgian Operational Research Society.* ORBEL 2017, Brussels, Belgium - February 02, 2017.

Data published online

* **3D Packing Problem with practical constraints:**
  Instance files for the BR instances modified to include extra constraints, such as loading priority and load balance, for example:
  - `http://dx.doi.org/10.17632/vcp6btht2h.1`

* **Instances for the extraction of the maximal object:**
  Files for the instances used to solve the problem of extracting the maximal object from a three-dimensional material:
  - `http://dx.doi.org/10.17632/phhzsbpkj9.1`



FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE
COMBINATORIAL OPTIMIZATION AND DECISION SUPPORT - CODES
Celestijnenlaan 200A box 2402
B-3001 Leuven
everton.fernandesdasilva@cs.kuleuven.be