

Optimal Packing of Irregular 3D Objects For Transportation and Disposal

by

Yinghui Zhao

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Civil Engineering

Waterloo, Ontario, Canada, 2021

©Yinghui Zhao 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner: Dr. Mani Golparvar-Fard
Associate Professor
Department of Civil and Environmental Engineering
University of Illinois at Urbana-Champaign

Supervisors: Dr. Carl T. Haas
Professor
Department of Civil and Environmental Engineering
University of Waterloo

Internal Members: Dr. Sriram Narasimhan
Professor
Department of Civil and Environmental Engineering
University of Waterloo

 Dr. Giovanni Cascante
Professor
Department of Civil and Environmental Engineering
University of Waterloo

Internal-external Member: Dr. Keith Hipel
Professor
Department of Systems Design Engineering
University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Parts of this thesis are based on the results of collaborative research and co-authored publications as detailed here:

Research presented in Chapters 2:

The manuscript is co-authored by Yinghui Zhao, Dr. Carl Haas, and Dr. Sriram Narasimhan. Yinghui Zhao and Dr. Carl Haas conceived the study. Yinghui Zhao was responsible for data collection, data analysis and interpretation, as well as drafting the manuscript. Dr. Carl Haas and Dr. Sriram Narasimhan provided guidance during each step of the study and critical revision on draft manuscripts.

Research presented in Chapter 3:

- **Zhao, Y., Rausch, C., & Haas, C. (2021).** Optimizing 3D irregular object packing from 3D scans using metaheuristics. *Advanced Engineering Informatics*, 47, 101234. DOI: <https://doi.org/10.1016/j.aei.2020.101234>

The research was conducted by Yinghui Zhao and Dr. Chris Rausch at the University of Waterloo under the supervision of Dr. Carl Haas. Yinghui Zhao was responsible for the conceptualization, methodology development, data collection and analysis, as well as drafting the original manuscript. Dr. Chris Rausch implemented the packing optimization algorithms in Rhinoceros® and contributed to the drafting and editing of the manuscript. Dr. Carl Haas provided guidance during each step of the research and provided feedback on draft manuscripts.

Research presented in Chapter 4:

- **Zhao, Y., Jung, Y., Haas, C., & Narasimhan, S. (2021).** Creating efficiency and mitigating risks in tactical work planning using virtual reality environments. *Automation in Construction*, Submitted in September 2021, (Under review).

The research was conducted by Yinghui Zhao and Yun-Ha Jung at the University of Waterloo under the supervision of Dr. Carl Haas and Dr. Sriram Narasimhan. Yinghui Zhao was responsible for the conceptualization, methodology development, data collection and analysis, as well as drafting the original manuscript. Yinghui Zhao and Yun-Ha Jung developed the algorithms and implemented the virtual reality planning environments and conducted the experiments. Dr. Carl Haas and Dr. Sriram Narasimhan provided guidance during each step of the study and critical revision on draft manuscripts.

Abstract

This research developed algorithms, platforms, and workflows that can optimize the packing of 3D irregular objects while guaranteeing an acceptable processing time for real-life problems, including but not limited to nuclear waste packing optimization. Many nuclear power plants (NPPs) are approaching their end of intended design life, and approximately half of existing NPPs will be shut down in the next two decades. Since decommissioning and demolition of these NPPs will lead to a significant increase in waste inventory, there is an escalating demand for technologies and processes that can efficiently manage the decommissioning and demolition (D&D) activities, especially optimal packing of NPP waste. To minimize the packing volume of NPP waste, the objective is to arrange irregular-shaped waste objects into one or a set of containers such that container volume utilization is maximized, or container size is minimized. Constraints also include weight and radiation limits per container imposed by transportation requirements and the waste acceptance requirements of storage facilities and repositories.

This problem falls under the theoretical realm of cutting and packing problems, precisely, the 3D irregular packing problem. Despite its broad applications and substantial potential, research on 3D irregular cutting and packing problems is still nascent, and largely absent in construction and civil engineering. Finding good solutions for real-life problems, such as the one mentioned above, through current approaches is computationally expensive and time-consuming. New algorithms and technologies, and processes are required.

This research adopted 3D scanning as a means of geometry acquisition of as-is 3D irregular objects (e.g., nuclear waste generated from decommissioning and demolition of nuclear power plants), and a metaheuristics-based packing algorithm is implemented to find good packing configurations. Given the inefficiency of fully autonomous packing algorithms, a virtual reality (VR) interactive platform allowing human intervention in the packing process was developed to decrease the time and computation power required, while potentially achieving better outcomes. The VR platform was created using the Unity® game engine and its physics engine to mimic real-world physics (e.g., gravity and collision). Validation in terms of feasibility, efficiency, and rationality of the presented algorithms and the VR platform is achieved through functional demonstration with case studies. Different optimal packing workflows were simulated and evaluated in the VR platform.

Together, these algorithms, the VR platform, and workflows form a rational and systematic framework to tackle the optimal packing of 3D irregular objects in civil engineering and construction. The overall framework presented in this research has been demonstrated to effectively provide packing configurations with higher packing efficiency in an adequate amount of time compared to conventional methods. The findings from this research can be applied to numerous construction and manufacturing activities, such as optimal packing of prefabricated construction assemblies, facility waste management, and 3D printing.

Acknowledgements

First and foremost, I would like to express my heartfelt appreciation to my supervisor, Dr. Carl Haas. He is an excellent supervisor, a visionary mentor, and a caring and supportive friend who I will always admire. I consider myself extremely fortunate to have worked with him and benefitted immensely from his words and deeds.

I would also like to express my gratitude to the members of my dissertation committee, Dr. Mani Golparvar-Fard from University of Illinois at Urbana-Champaign, Dr. Keith Hipel, Dr. Sriram Narasimhan, and Dr. Giovanni Cascante from the University of Waterloo, for devoting their time and expertise, as well as providing constructive feedback, to help me improve this dissertation.

I would like to express my gratitude to the Natural Sciences and Engineering Research Council of Canada for funding support provided through their Collaborative Research and Development and Alliance Grants programs, as well as to Ontario Power Generation (OPG) for providing matching funds. Additionally, I would like to thank OPG employees Alex Iliescu and Vasile Bostan for their technical assistance and constructive feedback throughout the development of this work.

My heartfelt appreciation goes out to my graduate classmates and friends: Chris, Ekin, JuHyeoung, Gabriel, Daniel, Christobal, Steven, Seokyoung, Mahdi, Ben, Lanlan, and Carol. Additionally, it was a pleasure to work with two outstanding cooperative students and research assistants: Yun-Ha and Emre.

Above all, I would like to express my gratitude to my family for their love, patience, unwavering support, and encouragement, all of which contributed to the completion of my dissertation. None of my accomplishments would have been possible without their understanding and commitment.

Table of Contents

Examining Committee Membership	ii
Author's Declaration.....	iii
Statement of Contributions	iv
Abstract.....	vi
Acknowledgements.....	viii
List of Figures	xii
List of Tables	xiv
CHAPTER 1 INTRODUCTION	1
<i>1.1 Background.....</i>	<i>1</i>
<i>1.2 Research Scope.....</i>	<i>3</i>
<i>1.3 Research Objectives.....</i>	<i>3</i>
<i>1.4 Research Methodology</i>	<i>4</i>
<i>1.5 Structure of The Dissertation.....</i>	<i>5</i>
CHAPTER 2 A SURVEY OF THE LITERATURE ON 3D IRREGULAR C&P PROBLEMS.....	7
<i>2.1 Summary</i>	<i>7</i>
<i>2.2 Introduction</i>	<i>7</i>
<i>2.3 Background.....</i>	<i>10</i>
<i>2.4 Variables.....</i>	<i>13</i>
<i>2.5 Geometry.....</i>	<i>13</i>
2.5.1. Geometry Representation.....	13
2.5.2. Geometry Complexity	16
<i>2.6 Constraints.....</i>	<i>19</i>
<i>2.7 Objectives</i>	<i>20</i>

2.8 <i>Optimization Search Techniques</i>	21
2.8.1. Constructive Heuristics	21
2.8.2. Metaheuristics.....	24
2.8.3. Mathematical Programming	27
2.9 <i>Discussion</i>	30
2.9.1. Trade-offs	30
2.9.2. Research Gaps and Future Research Opportunities	31
2.10 <i>Conclusion</i>	33
CHAPTER 3 OPTIMIZING 3D IRREGULAR OBJECT PACKING FROM 3D SCANS USING METAHEURISTICS	35
3.1 <i>Summary</i>	35
3.2 <i>Introduction</i>	35
3.3 <i>Related Work</i>	37
3.3.1. Geometry Representation	37
3.3.2. Mathematical Modeling-based Approaches	38
3.3.3. Heuristics.....	39
3.3.4. Partitioning and Packing Problem	41
3.4 <i>Proposed Methodology</i>	42
3.4.1. 3D Imaging.....	43
3.4.2. Irregular Data Processing	43
3.4.3. Packing Optimization Process	45
3.4.4. Initialization.....	45
3.4.5. Fitness Function.....	46
3.4.6. Metaheuristics.....	50
3.5 <i>Experiments with Archetypical Situations</i>	51
3.5.1. Situation 1.....	51
3.5.2. Situation 2:.....	52
3.5.3. Situation 3:.....	54
3.5.4. Sensitivity Analysis	55
3.6 <i>Discussion</i>	58
3.7 <i>Conclusion and outlook</i>	61

CHAPTER 4 CREATING EFFICIENCY AND MITIGATING RISKS IN TACTICAL WORK	
PLANNING USING VIRTUAL REALITY ENVIRONMENTS.....	62
4.1 Summary	62
4.2 Introduction	62
4.3 Literature Review.....	64
4.4 Methodology	66
4.5 Virtual Packing Environments.....	71
4.5.1. Integrated Heuristic Autonomous Packing Algorithm.....	74
4.6 Results and Analysis	77
4.6.1. Exposure Time	78
4.6.2. Invalid Packing Configurations.....	79
4.6.3. Packing Efficiency	80
4.7 Conclusions, Limitations, and Outlook	82
CHAPTER 5 CONCLUSIONS.....	84
5.1 Summary and contributions	84
5.2 Limitations and future research directions.....	86
5.3 Publications	88
5.3.1. Peer-refereed journal articles	89
5.3.2. Peer-refereed conference papers	89
Bibliography	90
Appendices	105

List of Figures

Figure 1-1. Estimated annual radioactive waste from decommissioning in Canada, 2013–2100. Reprinted from “ Inventory of radioactive waste in Canada 2016” by Natural Resources Canada [17].	2
Figure 1-2. Thesis methodology and structure.....	4
Figure 2-1. (a) Equipment and piping as found inside a power plant (PC: nostalgic/Shutterstock.com) (b) Disassembled pipe components packed inside a container in a simulated environment.....	9
Figure 2-2. Cutting stock problem	10
Figure 2-3. Container loading problem	11
Figure 2-4. Point cloud of objects being packed into a container	15
Figure 2-5. Phi function between two circles.....	16
Figure 2-6. Examples of different geometry complexity levels: (a) Simple, (b) Intermediate, (c) High.	18
Figure 2-7. A packing configuration generated from BLF	23
Figure 2-8. Decomposition of a large component.....	33
Figure 3-1. Overall methodology for metaheuristic-based optimal packing of irregular objects into a container	42
Figure 3-2 Overall methodology for metaheuristic-based optimal packing of irregular objects into a container.....	42
Figure 3-3. Converting the point cloud of a dome-shaped object into a mesh. (a) Point cloud of the dome. (b) Conversion of the point cloud to a mesh. (c) Mesh after manual correction.....	45
Figure 3-4. Translation range of an object with respect to the container.....	46
Figure 3-5. Packing result of 2×2×2 small cubes inside a cube container. (a) Initial placement. (b) Packing result after 20 minutes	52
Figure 3-6. Optimum packing of 4 foam pieces.....	53
Figure 3-7. Packing result of 4 foam pieces into a container slightly bigger than the original rectangular block. (a) Random initial placement. (b) Packing result after 45 minutes.....	53
Figure 3-8. Packing result of 4 foam pieces into a container slightly bigger than the original rectangular block. (a) Initial placement that is close to feasible solutions. (b) Packing result after 20 minutes	54

Figure 3-9. Packing result of real-life as-is objects into a rectangular container. (a) 8 as-is objects (b) Packing result after 45 minutes.....	55
Figure 3-10. Workflow of the selection algorithm	60
Figure 4-1: Workflows for different packing scenarios: (a) conventional approaches. (b) VR planning approaches	69
Figure 4-2. Hardware setting of the VR environments. (a) HTC VIVE® VR kit. (b) Hardware setup for the VR environments	72
Figure 4-3: Screenshots of the packing environment for (a) sequential packing scenario, (b) backhoe loader packing scenario, (c) human-guided packing scenario, and (d) automated packing scenario...	74
Figure 4-4. The pseudo-code for BLF	76
Figure 4-5: The GA-based autonomous packing algorithm	77
Figure 4-6: Average activity times for different scenarios (unit: s)	79
Figure 4-7: Packing efficiencies achieved in different scenarios	81

List of Tables

Table 2-1. Publications based on constructive heuristics.....	22
Table 2-2. Publications based on metaheuristics	25
Table 2-3. Publication based on mathematical programming.....	28
Table 3-1: Packing results of the 8 objects in situation 3 with 5 random starting positions.....	56
Table 3-2: Summary of packing results of 5 random subsets with five random starting positions	57
Table 4-1: Number of invalid packing configurations generated in different scenarios.....	80
Table 4-2: Results from the Games-Howell post hoc test.....	81

CHAPTER 1

INTRODUCTION

1.1 Background

Cutting and packing (C&P) problems have been studied in different disciplines such as mathematics [1,2], computer science [3,4], operations research [5–7] and engineering [8,9]. Optimal packing of irregular 3D objects, commonly referred to as the 3D irregular C&P problem is arguably the most challenging sub-problem in this area due to its computational complexity caused by geometry irregularity. The 3D irregular C&P problem involves arranging irregular-shaped objects into one or a set of containers to optimize one or multiple objectives, such as maximizing the packing efficiency or minimizing the container's volume. There is a growing interest in the 3D irregular C&P problem because of its broad applications and potential impacts in a multitude of industries. However, its applications and potential impacts in the civil engineering and construction industry are still overlooked.

As a matter of fact, the 3D irregular C&P problem can generally be applied both to traditional applications such as improving transport and storage efficiency of building parts or pre-fabricated construction assemblies as well as emerging applications in civil engineering such as 3D printing in construction and facility waste management, especially during the decommissioning and demolition (D&D) of nuclear power plants (NPPs) [10].

Globally, there are currently 443 nuclear reactors in operation as of May 13, 2021. 67.5% of the current operational nuclear reactors are older than 30 years, and around 25% are older than 40. According to International Energy Agency, around one-quarter of the current nuclear capacity in advanced economies is set to be shut down by 2025, and around 200 commercial reactors will be shut down by 2040 [11,12]. The decommissioning and demolition of NPPs often involves activities associated with the removal of fuel, safe storage, decontaminating structures and components, demolition of the building, and management of resulting waste [13]. The decommissioning of nuclear facilities will create substantial amounts of nuclear waste. Figure 1-1 depicts the projected annual NPP decommissioning waste volumes, a significant contributor to rapidly increasing nuclear waste inventory, in Canada through 2100. Similarly, according to a conservative estimate by the International Atomic Energy Agency, Europe's power reactor fleet alone may generate at least 1.4

million m³ of low- and intermediate-level waste from decommissioning [14]. The ability to efficiently and safely treat and dispose of radioactive materials has become an essential prerequisite for the nuclear facility D&D process [15]. Given the limited capacity in storage facilities and the cost of waste containers, optimal packing of the nuclear waste could potentially save millions of dollars in D&D costs. Given that a large proportion of decommissioning waste is highly heterogeneous in size and shape of individual components [16], research on 3D irregular C&P problems is required.

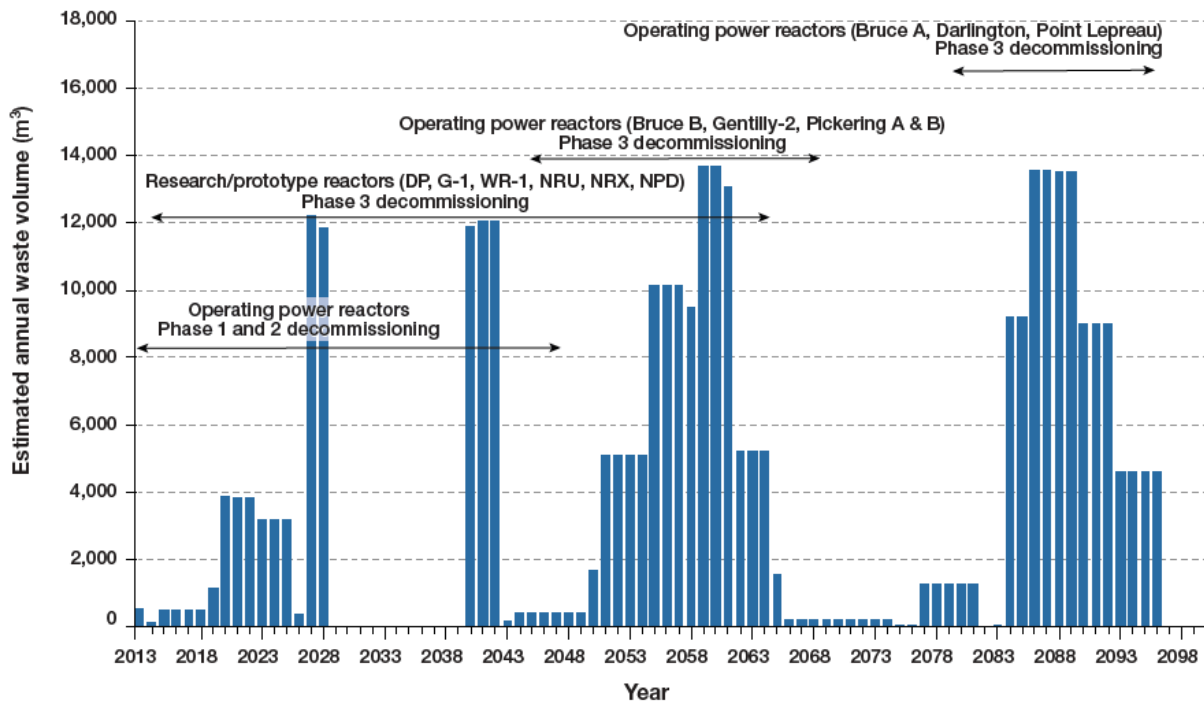


Figure 1-1. Estimated annual radioactive waste from decommissioning in Canada, 2013–2100.

Reprinted from “Inventory of radioactive waste in Canada 2016” by Natural Resources Canada [17].

However, research on 3D irregular C&P problems is still nascent despite its broad applications and substantial potential. The primary reason is that the 3D irregular C&P problem is known to be NP-hard, i.e., the expected time to find an optimal solution increases exponentially as a function of the number of inputs [18]. Researchers have proposed different approaches, including constructive heuristics, metaheuristics, and mathematical programming. Nonetheless, none of the existing algorithms for the 3D irregular C&P problem can find a globally optimal solution in polynomial time [19].

This research takes the optimal packing of nuclear waste in D&D of NPPs as an application problem, and aims to explore adaptive packing algorithms, tools, and workflows that generate good quality packing results within acceptable computational complexity and processing time to deal with real-world implementations in the civil engineering and construction industries.

1.2 Research Scope

While optimal packing of 3D irregular objects has broad applications and potential impacts in various industries, its application in civil engineering is not yet explored. This research mainly focuses on nuclear waste packing optimization in facility D&D for verification and validation purposes. The algorithms, tools, and workflows developed based on this application can be highly customizable and adapted to numerous construction and manufacturing activities, which will lead to improvement in efficiency and reduction in risk and cost.

Another fundamental premise of this research, which defines and limits its scope and methodology, is that the shape of the irregular objects created during D&D are most effectively represented by 3D scans. Therefore, it is assumed that 3D point cloud or mesh representations will be used for the objects rather than trying to fit 3D irregular objects into gross parametric approximations, such as bounding boxes or cylinders[20][21]. This creates advantages and new opportunities for fitting and packing efficiencies. For example, Hur et al. [22] compared the bounding box approach and the voxelization approach and figured out the latter obtained a better packing efficiency.

1.3 Research Objectives

The overarching goal of this research is to develop algorithms, tools, and workflows to optimize the packing of 3D irregular objects in an acceptable processing time while guaranteeing lower risks imposed on human operators and reduction in rework and cost. Consistent with the overall goal, specific objectives are identified:

- Develop algorithms and a methodology that enables automated packing optimization of as-is 3D irregular objects (e.g., nuclear waste generated from D&D of NPPs).
- Develop a general visualization interface to show the packing configuration and evaluation.
- Develop an interactive packing platform that enables manual intervention into the packing process to achieve better adaptability and operability.

- Identify workflows or framework that release human operators from complicated and hazardous working environments.

1.4 Research Methodology

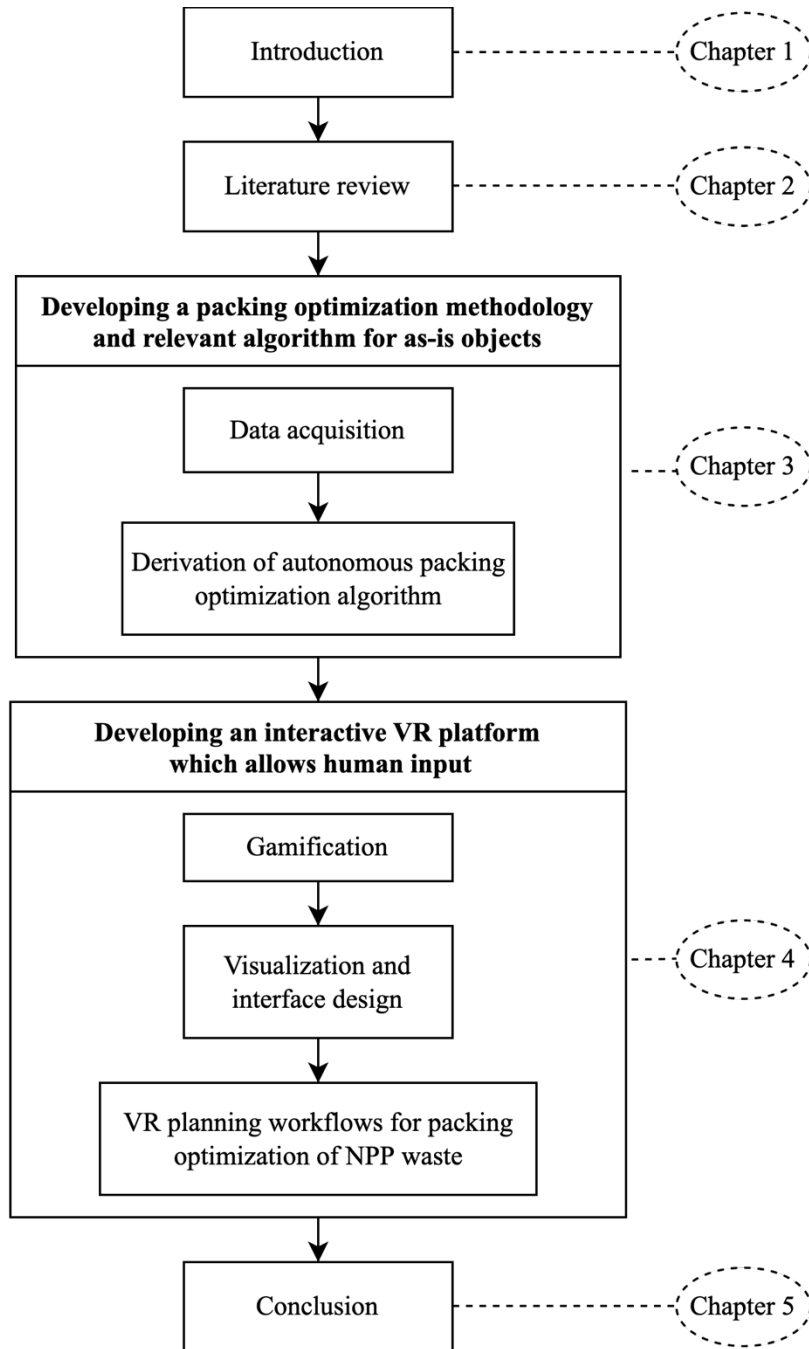


Figure 1-2. Thesis methodology and structure

Figure 1-2 illustrates the methodology used in this thesis to optimize the packing of 3D irregular objects. A comprehensive review of the existing literature on 3D irregular C&P problems in various industries is conducted, along with the identification of knowledge gaps and future research opportunities in civil engineering applications. Then, a framework for optimal packing of as-is objects is presented, which is comprised of data acquisition and the derivation of a packing optimization algorithm. After that, to mitigate the inefficiency of fully autonomous algorithms, a customizable interactive virtual reality (VR) packing platform is developed that allows for human input during the packing process, thereby speeding up the packing process and improving the packing results. Two workflows based on the VR packing platform are identified and compared to conventional packing methods to evaluate the performance of and validate the proposed methods. Finally, conclusions, limitations, and future research directions are discussed.

1.5 Structure of The Dissertation

The dissertation is organized and presented in five chapters:

- **Chapter 1: Introduction.** This chapter introduces the 3D irregular C&P problem and its applications in construction, especially in optimal packing of nuclear waste, and states the motivation for this research. Research scope and objectives are identified and outlined. Research methodology is introduced.
- **Chapter 2: A survey of the literature on 3D irregular C&P problems.** To contextualize the research objectives, this chapter provides a detailed literature review on 3D irregular C&P problems. The major components of the 3D irregular C&P problem are identified. Different existing approaches are categorized and evaluated to frame the knowledge gaps, facilitate understanding in civil engineering, and provide insight into future research opportunities.
- **Chapter 3: Optimizing 3D irregular object packing from 3D scans using metaheuristics.** This chapter presents a 3D packing optimization methodology and relevant algorithms for as-is 3D irregular objects. The methodology begins with capturing the initial as-is 3D shape data of 3D irregular objects, followed by a metaheuristic-based algorithm for packing optimization. This approach keeps the as-is shape of the irregular objects, which can provide more accurate and potentially denser packing configurations.
- **Chapter 4: VR platform for 3D irregular packing problem.** This chapter focuses on developing an interactive VR platform which allows 3D objects to be packed into a pre-

specified container with human input while optimizing multiple objectives. Furthermore, different workflows for packing optimization of NPP waste are identified. VR-enhanced packing environments are developed to optimize nuclear waste packing and minimize risk due to radiation exposure associated with commonly employed manual packing approaches.

- **Chapter 5: Conclusions.** This chapter summarizes the findings and contributions that can be drawn from the research. Limitations of the current methodologies are discussed. Future research opportunities are proposed.

CHAPTER 2

A SURVEY OF THE LITERATURE ON 3D IRREGULAR C&P PROBLEMS¹

2.1 Summary

Optimal packing of 3D irregular objects has a wide range of applications in 3D printing, manufacturing, and any other industry that involves layout design, transportation, and storage. This type of problem is known as 3D irregular cutting and packing (C&P) problems in many disciplines. There is an increasing need to address 3D irregular C&P problems in civil engineering applications such as optimal packing of prefabricated construction assemblies and construction waste, and nuclear waste from nuclear power plant (NPP) decommissioning and demolition (D&D). Because of the inter-disciplinary nature of this research and the wide range of end applications, literature landscape on 3D irregular C&P problems is fragmented. The review papers on this topic are either dated or are focused on a specific industry and methodology. None have reviewed this area from the standpoint of civil engineering applications, even though there is enormous potential to develop new methods and translate existing methodologies and tools from other fields to specific applications in civil engineering. Hence, this survey aims to address this crucial gap and presents an up-to-date and comprehensive review of 3D irregular C&P problems drawn from several disciplines, while also connecting it with the potential to address unique challenges in civil engineering problems. Specifically, this survey identifies and reviews the major components of 3D irregular C&P problems, provides insight into the application of 3D irregular C&P problems in civil engineering and identifies future research opportunities. It also identifies the specific knowledge gaps addressed by this thesis.

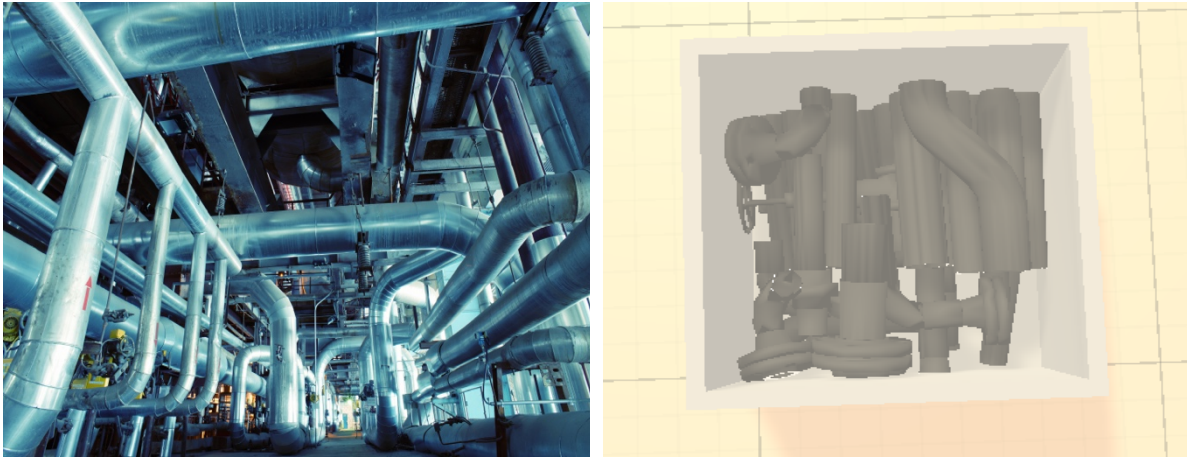
2.2 Introduction

Three-dimensional irregular C&P problems involve arranging a set of 3D irregular objects into a set of containers in order to optimize one or more objectives, such as maximizing container space utilization. Three-dimensional irregular C&P problems have piqued the interest of the scientific and practitioner communities due to the wide range of applications they can support and the potential impact they can have. Additive manufacturing, also known as rapid prototyping or 3D printing, is a

¹ This chapter is adapted from Zhao, Y., Haas, C., & Narasimhan, S. (2021). A survey of the literature on 3d irregular C&P problems. (In preparation)

popular application that refers to technologies capable of producing 3D shapes by progressively adding layers. Optimizing the arrangement of the objects within the building space prior to printing, as well as simultaneously creating multiple objects in one printing time, can help to speed up production and cut costs. Commodity and material packing, transportation and storage, and component layout design for compact volumes such as computer hardware, satellites, submarines, or high-speed trains are among the other applications.

Three-dimensional irregular C&P problems have a lot of potential in solving unique challenges in civil engineering and construction. For example, there is an urgent need to develop methods to optimize the packing volume of nuclear waste generated from D&D activities since 25% of existing nuclear capacity in advanced economies expected to be shut down by 2025 [12]. The goal of optimizing nuclear waste packing volume is to disassemble large components and structures in nuclear power plants and arrange the resulting 3D irregular-shaped nuclear waste into containers in such a way that container space utilization is maximized, or container size is minimized. Given the limited capacity of storage facilities and the rising cost of waste management, optimal nuclear waste packing could potentially save millions of dollars in D&D costs. Figure 2-1a depicts equipment and piping as found inside a power plant, while Figure 2-1b depicts a nuclear waste packing configuration in a simulated environment. 3D irregular packing problems are also relevant in other civil engineering applications, including the placement of construction materials on construction sites [23], improving the transport efficiency of building parts, or prefabricated construction assemblies [24], as well as some emerging civil applications such as 3D printing in construction [25] and dry stacking [26]. Unfortunately, despite their significant potential impact, research and practical applications of 3D irregular C&P problems in the context of civil engineering applications is still extremely limited.



(a)

(b)

Figure 2-1. (a) Equipment and piping as found inside a power plant (PC: nostalg6ie/Shutterstock.com) (b) Disassembled pipe components packed inside a container in a simulated environment.

As mentioned previously, 3D irregular C&P problems have been studied in many disciplines (other than civil engineering) and industries due to their potential to solve challenging optimization scenarios. For the same reason, literature landscape on 3D irregular C&P problems is fragmented. Oh et al. and Arajo et al. investigated 3D irregular C&P problems in the additive manufacturing industry [27,28]. Leao et al. conducted a review of the mathematical models proposed for 3D irregular C&P problems [1]. These reviews only cover a small portion of the state-of-the-art in 3D irregular C&P problems. Nearly 20 years ago, Cagan et al. provided an excellent overview of 3D irregular C&P problems in component layout and packaging applications [29]. However, new approaches to problem solving have been proposed over the last two decades and with the recent advancements in this field, a literature survey of 3D irregular C&P problems covering more recent developments with emphasis on civil engineering applications is required

The purpose of this chapter is to summarize the current state of the art in 3D irregular C&P problems and to aid in the comprehension of 3D irregular C&P problems in civil engineering. To do this, relevant articles from 1995 to 2021 are reviewed, with an emphasis on features critical to civil engineering. Due to the limited literature specific to civil engineering, authors have instead provided connections between the relevant literature and potential problems arising in civil engineering where such methods could be readily applied. Additionally, the solution approaches described in these publications are examined and classified to assist readers in better comprehending state-of-the-art

achievements. The research gaps and future research prospects are highlighted to encourage additional fruitful research in this field.

2.3 Background

Three-dimensional C&P problems are included in the broader category of C&P problems, which encompass two research areas that have been found later to have strong relationships: (1) cutting problems and (2) packing problems [6,30]. Cutting problems, such as cutting stock and trim loss problems in manufacturing (steel, textiles) depicted in Figure 2-2, involve dividing large materials into smaller pieces while minimizing material waste. Packing problems, such as the container loading shown in Figure 2-3, and component layout problems, on the other hand, entail packing small objects into large containers as densely as possible or using as few containers as possible. The underlying structure of the two problems is the same: the arrangement of small objects (e.g., nuclear waste, pieces of material) into large objects (e.g., containers and large raw-materials) while adhering to the constraints that the small objects must not overlap and must be entirely contained within the large containers in order to optimize one or more objectives, for example, minimizing waste material in trim loss problems or maximizing packing efficiency in container loading problems [30].

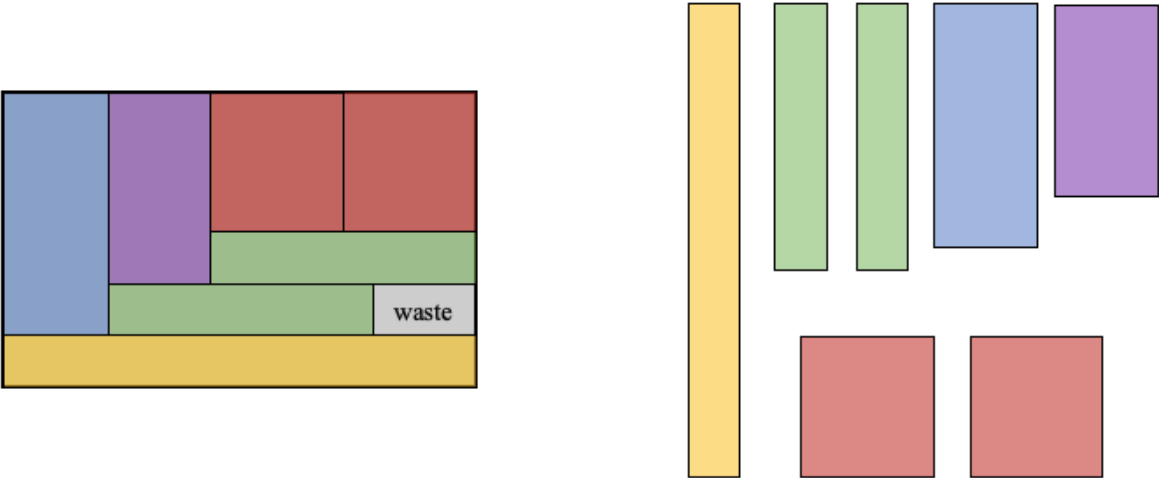


Figure 2-2. Cutting stock problem

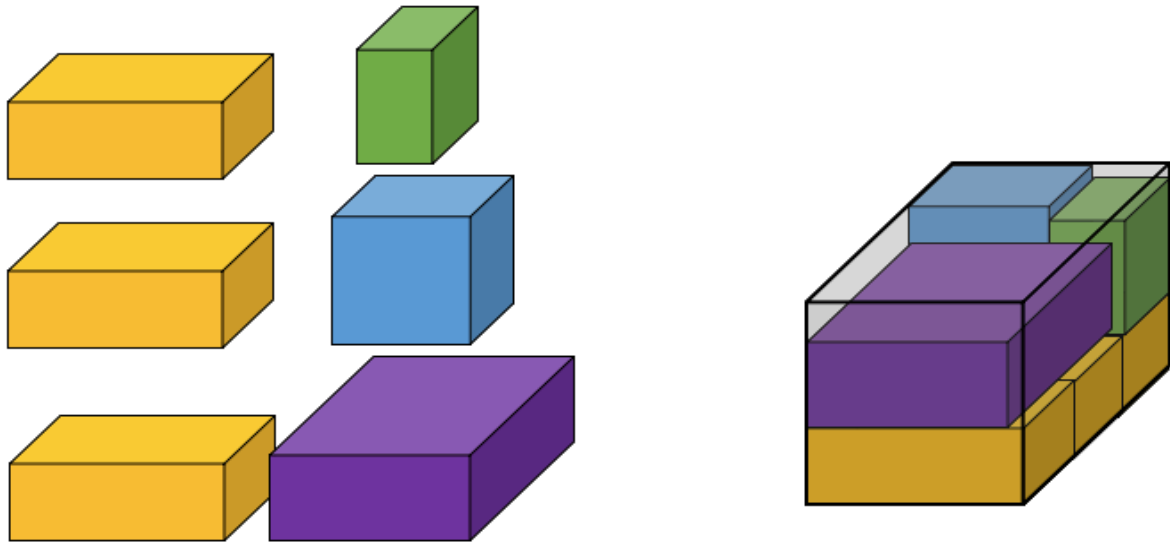


Figure 2-3. Container loading problem

While C&P problem is an umbrella term encompassing problems with essentially identical logical structure as previously outlined, Dyckhoff proposed four criteria for categorizing C&P problems in a systematic manner: dimension, kind of assignment, and assortment of large and tiny objects [30]. Wäscher et al. extended Dyckhoff's typology by including a fifth criterion: the shape of small objects [6]. Combining the distinct types of dimensionality, assignment, assortment and shape of objects can result in a variety of C&P problems.

This thesis focuses on three-dimensional irregular C&P problems characterized by dimensionality and shape of small objects. Dimensionality refers to the minimal number of dimensions necessary to accurately represent the required layouts[6]. When cutting stocked metal bars, pipelines, and steel rails to various specified lengths while minimizing material waste, one-dimensional C&P problems arise. [31–33]. Two-dimensional C&P problems are useful in industries such as shoe, apparel, and furniture manufacturing, where components must be cut from large sheets of material, as well as in construction applications such as site layout design and architectural floor plan layout design [34–36]. Site facilities such as material storage areas, tool storage areas, operation plants, and other site accessories can be logically allocated to maximize space utilization and accessibility while minimizing overall operation costs. The optimization of architectural floor plan layouts entails determining feasible locations and dimensions for a set of rooms that satisfy design requirements and maximize design quality in terms of design preference. Three-dimensional C&P problems include,

but are not limited to, vehicle loading, pallet or container loading, and 3D component layout problems [37–39].

The shape of small objects distinguishes between C&P problems involving regular-shaped objects and irregular-shaped objects. Bin packing problems, where rectangular-shaped boxes need to be packed, are 3D regular C&P problems; A cutting stock problem, where irregular pieces must be produced from a large raw-material sheet while minimizing the trim loss, is a two-dimensional irregular C&P problem. In contrast to regular shapes (such as rectangular, cylindrical, prismatic, etc.), an irregular form cannot be parametrically specified using a shape type and a limited set of parameter primitives. This results in a different set of geometric representations and methodologies for C&P problem of irregular objects, which will be discussed in greater detail later in this chapter.

The structure of 3D irregular C&P problems can be summarized mathematically as follows. We have a set of containers denoted by C and a set of n objects denoted by O (the i^{th} object is denoted by $O_i, i = 1, \dots, n$). Set $O_i \oplus v_i = \{o_i + v_i | o_i \in O_i\}$ is the Minkowski sum of O_i and a vector v_i , which can be used to define object O_i is translated by v_i , where \oplus is the Minkowski sum operator and o_i is a point in object O_i [1]. By denoting $\theta_i = (\theta_{ix}, \theta_{iy}, \theta_{iz})$ the rotation of object O_i about the x, y, and z axes, the rotation of object O_i can be defined as $O_i(\theta_i) = R_{iz}R_{iy}R_{ix}O_i$, where

$$\begin{aligned}
 R_{ix} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_{ix} & -\sin\theta_{ix} \\ 0 & \sin\theta_{ix} & \cos\theta_{ix} \end{bmatrix} \\
 R_{iy} &= \begin{bmatrix} \cos\theta_{iy} & 0 & \sin\theta_{iy} \\ 0 & 1 & 0 \\ -\sin\theta_{iy} & 0 & \cos\theta_{iy} \end{bmatrix} \\
 R_{iz} &= \begin{bmatrix} \cos\theta_{iz} & -\sin\theta_{iz} & 0 \\ \sin\theta_{iz} & \cos\theta_{iz} & 0 \\ 0 & 0 & 1 \end{bmatrix}
 \end{aligned} \tag{2-1}$$

$f(v_i, \theta_i)$ is the objective function of the problem, the expression of which takes different forms depending on the specifics of each problem. As per the discussion above, the mathematical formulation of 3D irregular C&P problems can be defined as follows:

$$\begin{aligned}
 &\text{minimize } f(v_i, \theta_i) \\
 &\text{s.t. } O_i(\theta_i) \oplus v_i \subseteq C
 \end{aligned} \tag{2-2}$$

$$(O_i(\theta_i) \oplus v_i) \cap (O_j(\theta_j) \oplus v_j) = \emptyset$$

$$i = 1, \dots, n; j = 1, \dots, n; i \neq j$$

Informed by [29], five critical components for addressing a 3D irregular C&P problem are identified: variables, constraints, geometry representations, objectives, and optimization search techniques. The next sections address the five components in detail to give an in-depth and reasonably thorough assessment of the existing literature on 3D irregular C&P problems.

2.4 Variables

The positions and orientations of objects are variables in 3D irregular C&P problems that must be resolved in order to optimize certain objectives. The objects' positions and orientations are frequently represented by translation and rotation vectors in the Cartesian coordinate system. The six degrees of freedom for object O_i can be denoted by $(v_{ix}, v_{iy}, v_{iz}, \theta_{ix}, \theta_{iy}, \theta_{iz})$, where $v_i = (v_{ix}, v_{iy}, v_{iz})$ is the vector of translation and $\theta_i = (\theta_{ix}, \theta_{iy}, \theta_{iz})$ is the object's rotation about the x, y, and z axes, respectively. Scale factor is also a variable in approaches that involve shrinking and growing the size of objects [40].

2.5 Geometry

2.5.1. Geometry Representation

In contrast to regular shapes (rectangular, cylindrical, prismatic, etc.), and as noted earlier, irregular geometries cannot be parametrically defined using a shape type and a limited set of object-specific parameter primitives [41]. An irregular object can be represented in a variety of ways, which then leads to different optimization search techniques. As a result, the geometry representation is a critical step in the solution process. In 3D irregular C&P problems, there are primarily four different geometry representations used:

- Bounding volume representation, e.g., bounding box, bounding cylinder, bounding sphere
- Volumetric representation, e.g., voxel, octree, and sphere tree, 3D matrix representation
- Boundary representation, e.g., triangular surface mesh, point cloud
- Mathematical representation, e.g., Phi function

Bounding volume representation. The bounding volume representation employs a straightforward bounding shape to represent a complex irregular object. For instance, an object's

bounding box is a closed box that encompasses the entire object. Another popular bounding volume representation is the bounding cylinder. The majority of early research makes use of bounding volume representations to improve the efficiency of object collision detection. Bounding representations, on the other hand, have been shown to lead to inefficient solutions, as the hollow space between the bounding volumes and the objects is squandered [22]. As a result, the representation is being phased out in recent publications.

Volumetric representation. An irregular object can be discretized into a list of 3D volume elements. The 3D volume elements can commonly be represented by a list of identical cubical grids, in other words, voxels. The discretization process is called voxelization. Some approaches represent 3D irregular geometries using voxels [42–44]. Furthermore, the 3D volume elements can be represented by spheres or other shapes and are not necessarily identical. Cagan et al., Stefan and Paul presented approaches based on multi-resolution and/or hierarchical representations that involve dividing an object into various levels of resolutions such as Octree and sphere tree [45,46]. The precision of the representation is directly associated with the size of the voxels and levels of resolution. However, higher resolutions and smaller voxel grid sizes will increase the computation time. Some researchers also decompose 3D objects into clusters of mutually orthogonal parallelepipeds, pyramidal primitives, or Tetris-like items to lower the volume element counts [37,47,48]. It is worth noting that there is no voxelization into irregular-shaped volume elements.

Boundary representation. The boundary representation represents the enclosing surface of an object by vertices, edges, and faces. For example, a polygon can be represented by a list of vertices, assuming straight lines and flat surfaces connecting adjacent vertices. One popular boundary representation for 3D irregular C&P problems is the triangular mesh. The idea is to tile the enclosing surface of the 3D geometry into connecting triangle faces. The most applied mesh file format is STL [3]. It is widely supported in the rapid prototyping and computer-aided design industry for synthetic models; it stores the vertices and unit normal vector of triangular faces. Another boundary representation applied in the literature is a point cloud. A point cloud is a set of points on an object's enclosing surface, usually obtained as the output of 3D scanning processes. It has wide applications in the construction industry as well as robotics. For 3D irregular C&P problems involving manufactured or as-is objects, the point cloud is an excellent way to represent 3D geometry information. The precision of the point cloud depends on the density of the points. Figure 2-4 shows a packing configuration represented by point cloud.

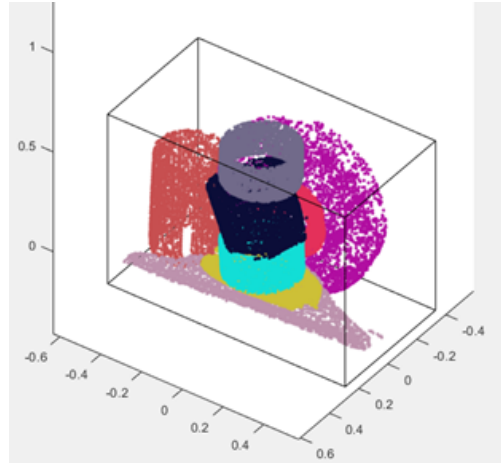
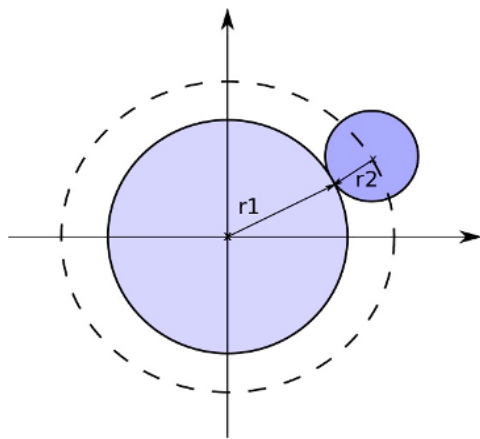


Figure 2-4. Point cloud of objects being packed into a container

Mathematical representation. As in the 3D C&P problems, researchers made efforts to formulate 3D irregular C&P problems mathematically. To this end, a mathematical representation of an object must be defined. In this regard, the concept of phi function was conceived and presented in [49]. Chernov et al. later define the class of general shapes permissible for phi function modeling scheme as phi-objects in 2021 [50]. A phi-object should have interior and boundary and should not have self-intersection along their boundary. 3D shapes like spheres, parallelepipeds, cylinders, or convex polyhedra, are called primary phi-objects. A primary phi-object is described by its type and its parameters, e.g., a sphere can be defined as a pair (S, r) , where S refers to the type sphere, and r is the radius. A convex polyhedron can be specified by its type K and a set of triples that define a set of half-spaces. More complex shapes are composed by forming unions and intersections of primary objects. A phi-function that formalizes the relevant position of two objects is the collision detection method between phi-objects. Let A and B be objects with placement parameters $u_A = (v_A, \theta_A)$ and $u_B = (v_B, \theta_B)$, where v_A and θ_A are the translation and rotation vectors of object A , and v_B and θ_B are the translation and rotation vectors of object B . Phi-function $\varphi^{AB}(u_A, u_B)$ is a continuous and everywhere defined function for object A and B , if:

- Non-collision : $\varphi^{AB}(u_A, u_B) > 0$
- Touching: $\varphi^{AB}(u_A, u_B) = 0$
- Collision : $\varphi^{AB}(u_A, u_B) < 0$



$$\Phi^{CC} = (x_1 - x_2)^2 + (y_1 - y_2)^2 - (r_1 + r_2)^2$$

Figure 2-5. Phi function between two circles

The phi-functions for different shapes are constructed in [50]. The Phi-function for two circles is shown in Figure 2-5. Phi-function for 3D objects, including many operators, is analytically complicated even for polyhedra. Stoyan et al. further developed the quasi-phi-function, a variation of the phi-function, with the same concept yet including auxiliary variables, which is substantially simplified compared to the phi-function for some types of objects, especially polyhedra [51]. More details on the construction of the quasi-phi-function of convex and concave polyhedra can be found in [7,51]. Phi-function and quasi-phi-function provide a tool to formulate the 3D irregular C&P problem in a general mathematical programming way.

2.5.2. Geometry Complexity

Geometry is a critical feature when dealing with three-dimensional irregular C&P problems. For 3D regular C&P problems with regular geometries, such as boxes and spheres, the geometry complexity may appear simple. However, irregular geometries exhibit a great deal of variety and a range of geometric complexity. Different geometry complexity can have an effect on the solution approach and corresponding algorithmic complexity for 3D irregular C&P problems.

As the three-dimensional irregular C&P problem spans multiple disciplines, researchers have expressed concern about the lack of a widely used benchmark with representative geometries [3]. Nonetheless, researchers pursuing mathematical approaches are concentrating their efforts on less complex geometries (i.e., fewer vertices, fewer facets). In comparison, researchers in the additive

manufacturing industry argue that more complex, benchmark datasets with thousands of meshes should be established [3]. The reality is that researchers are unlikely to agree on a single universal benchmark anytime soon, and the benefit of a standard set of benchmark objects is still unproven. Considering the foregoing, it is necessary to establish a common categorization standard for both existing objects sets and those that will be created in the future across diverse disciplines.

Some preliminary efforts have been made to categorize the geometries of 3D C&P problems. Egeblad and Pisinger classified 3D regular C&P boxes into five types: flat, long, cubes, uniform, and diverse [52]. Araujo et al. propose a comparison metric for 3D irregular C&P problems that uses the surface, number of triangles, and volume to compare the sets of irregular-shaped objects used in the additive manufacturing industry [3]. The metric considers both the file size and the size of the models. However, because the metric relies on the STL format, which is not always the case for all geometries in 3D irregular C&P problems, their classification is limited to additive manufacturing. To categorize different 3D irregular geometries in terms of geometry complexity, a simple assessment system is proposed. The proposed assessment system focuses on essential geometry features of objects such as convexity, surface variation, and symmetry; thus, it does not require complicated calculations and is file format independent.

Geometry properties such as convexity can be used to characterize an object's geometry complexity. Araujo et al. suggested using the “spies” ratio, a complexity metric commonly used in manufacturing, to determine the geometry complexity of objects [28]. The “spies” ratio is calculated by dividing the volume of the geometry part by the volume of the primitive that contains it. Sukumar et al. [53] demonstrated through a series of studies that protrusions and holes enhance geometry complexity, correlating with the assumption that convexity impacts geometry complexity. Additionally, their investigation indicated that symmetry and surface variation have an effect on how complicated geometry is perceived. Pei et al. classified 3D printing items according to their symmetry and surface variation features [54]. In their study, objects with planar surfaces and multiple symmetry axes are referred to as simple objects, whereas objects with variations on the surface and little symmetry are referred to as complex objects.

A comparable technique is proposed here, incorporating the geometric properties of objects such as convexity, symmetry, and surface variation. A concave object with a variation in surface curvature and little symmetry is considered complex. A convex object with no variation in curvature and

multiple symmetry axes is considered simple. In other words, to build up the scheme, three conditions are considered:

- The object is concave.
- The object has less or no symmetry axis.
- The object possesses variations of curvature on the surface.

Figure 2-6 depicts several examples of geometric complexity at various degrees. An object with high geometry complexity is the one that satisfies all of the conditions, such as the concrete rubble from D&D processes depicted in Figure 2-6c. The short section of the pipe in Figure 2-6b is deemed to be of intermediate complexity because it satisfies two of the conditions. Simple objects are those that satisfy a single or no condition, such as a cube or an ellipse. An analogical situation in civil engineering is the classification of aggregates, which considers elongation, flatness, and surface texture. The geometry complexity of a collection of objects is considered to depend on the complexity of the most complicated object.

The 3D irregular objects that are being addressed in civil engineering applications frequently lack corresponding synthetic models, and their geometry information must first be acquired using a variety of 3D imaging methods, such as 3D laser scanners, range cameras, photogrammetry, and structured light scanners, and then converted to the aforementioned geometry representations. The target objects, particularly the building waste generated during the D&D process, frequently exhibits intermediate to high degrees of geometry complexity.

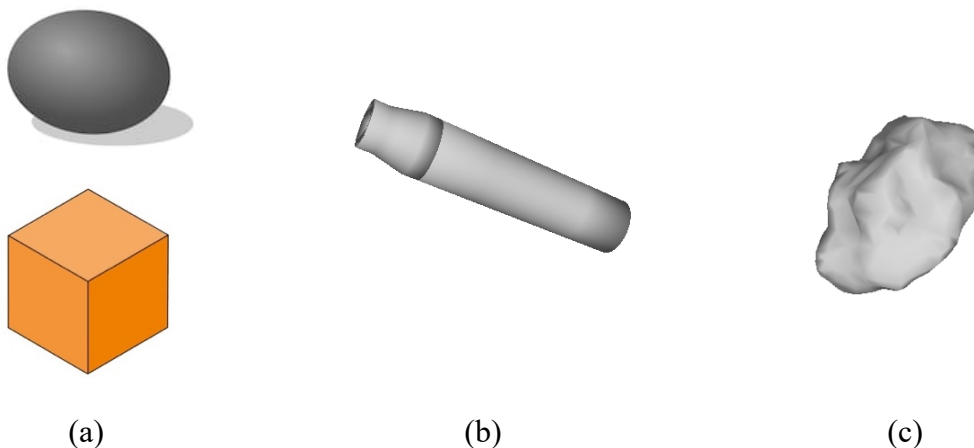


Figure 2-6. Examples of different geometry complexity levels: (a) Simple, (b) Intermediate, (c) High.

2.6 Constraints

For C&P problems, there are two primary constraints: (1) non-interference, and (2) containment. The non-interference constraint refers to the requirement that objects in the packed state be collision-free. The constraint on containment requires that the items be totally contained within the container. These two fundamental constraints are independent of the task at hand and focus exclusively on the geometry element; consequently, they are referred to as geometry constraints in this thesis.

With the growth of research in 3D irregular C&P problems, researchers have encountered more constraints in a variety of practical problems; for example, Bortfeldt and Wäscher summarized practically relevant constraints that can occur when loading containers [55]. Similarly, for 3D irregular C&P problems in civil engineering, certain typical constraints must be considered.

- **Stability or balance constraint:** The majority of existing approaches pack objects into containers without taking gravity or friction into account [56]. However, for real-world applications, real-world physics must be considered. Beyond the geometric assignment of a container, stability is also a critical issue. For instance, the container and the packed objects must remain stable during transportation and disposal of construction waste. Wang and Hauser determine whether or not each packed object is in static equilibrium under the influence of gravity and friction [56]. Wang et al. limit the total system's center of gravity point error to a specified allowable value [57]. Egblade et al. ensure that objects are adequately supported by limiting the height difference between the region beneath the object to a specified value [58]. Game engines and physics engines are effective methods for simulating the physics of packing configurations.
- **Manipulation feasibility:** This requirement requires that the packing configurations recommended by packing algorithms be viable and repeatable by people or robots in the actual world. As self-evident as it may appear, most publications overlook the limitation. Wang and Hauser addressed the challenge of completely automated object packing in warehouses utilizing robotic arms [56]. According to their analysis, manipulation feasibility necessitates the existence of a feasible robot motion capable of loading the object into the desired location. Feasibility checks are carried out on potential object transformations to ensure that they satisfy the manipulation feasibility constraint.
- **Positioning constraint:** This constraint restricts the object's placement relative to other objects or the container. For example, when transporting prefabricated construction assemblies,

specific subsets of the assemblies may need to be packed closely together to facilitate subsequent assembly; In the Saturn automobile engine compartment layout challenge mentioned in [45], the positioning constraint requires that all fluid reservoirs be situated near the engine's upright; in Jang and Rhee's packing problem for an ocean space submersible boat, the strobe must be located near the camera [59].

- Capacity constraints: This kind of constraint encompasses constraints on the container's non-geometrical capacity. For instance, capacity restrictions in nuclear power plant waste packing optimization include the container's weight limit during transit and the radiation limit imposed by nuclear waste storage facilities.

These additional restrictions are highly dependent on the problem; thus, they are classified in this thesis as problem-dependent constraints. Applying these practical restrictions adds another layer of complexity to the model's equations and may have an effect on the final solution strategy. Constraints can be handled in two ways: (1) by incorporating them as penalty terms in the objective function of for example genetic algorithms [8,10,60,61], or (2) by imposing them via continual constraint violation checking and restricting item placement to viable areas [56,62].

2.7 Objectives

Constraints impose restrictions on the optimization problem, while objectives represent the goals to be maximized or minimized in an optimization problem. The common objective of 3D irregular C&P problems is to increase space utilization inside the containers, which usually implies a decrease in cost for many industries, e.g., warehousing and transportation. Wascher et al. outlined two major scenarios for space maximization in C&P problems: output maximization and input minimization [6]. These two scenarios also apply to irregular C&P situations in three dimensions. The purpose of output maximization is to pack as many things as possible into a single container or a specified collection of containers [10,37,63–67]. Minimizing wasted space in a container is considered a subset of output minimization. The goal of input minimization is to group a given set of items into the fewest available containers. When dealing with problems requiring the allocation of objects to a single container, input minimization refers to minimizing the size of the container employed [68–70].

In civil engineering applications, the primary objective of space utilization should be taken into account. Apart from that, additional behavioral objectives should be investigated. Behavioral objectives define the desired behavior and quality of the proposed packing optimization approaches

and resulting packing configurations. For example, the computation time required to generate packing configurations must be minimized, particularly for applications that require real-time packing configuration generation, such as hazardous material packing optimization. Besides civil engineering, in the additive manufacturing industry, behavioral objectives for 3D irregular C&P problems may include, but are not limited to, minimizing the volume of supporting material and achieving a higher surface quality [3]. 3D irregular C&P problems, such as vessel layout design for satellites and submarines, necessitate the minimization of gravity center position deviation [57,59].

While the objectives of 3D irregular C&P problems vary, they are frequently expressed as objective functions, the values of which are used to rate probable packing layout solutions. The fundamental strategy for multi-objective optimization of the three-dimensional irregular C&P problem is to combine all objectives into a fitness function using the weighted sum [71]. Constraints, as discussed in Section 2.6, can be treated as penalty terms in the objective function. This type of objective function is straightforward to implement. However, the assignment of weights can significantly affect the performance of the approach. Inappropriate weighting factors may result in suboptimal, if not impossible, results. Wu et al. propose the use of Pareto boundaries for each objective to facilitate the determination of the superior solution [71]. Fakoor et al. propose using multi-objective optimization methods based on the Pareto frontier, dubbed non-dominant sorting genetic algorithm II (NSGA II), to address layout challenges for spaceship components [8].

2.8 Optimization Search Techniques

It is well-known that the 3D irregular C&P problems are NP-hard. In other words, the expected time required to find a global optimal solution increases exponentially as the number of inputs increases [18]. Three distinct search techniques have been developed to find solutions that are close to optimal or locally optimal.

2.8.1. Constructive Heuristics

Heuristics are rules of thumb for guiding, discovering, and revealing plausible but not necessarily correct solutions to problems [72]. A constructive heuristic is a low-level heuristic that incrementally builds a complete solution from scratch for a particular class of problems. Typically, constructive heuristics are problem dependent. Various constructive heuristics have been developed specifically for 3D irregular C&P problems. Table 2-1 summarizes publications that used constructive heuristics.

Table 2-1. Publications based on constructive heuristics

Number	Publication	Constraint type	Objective type	geometry complexity	Geometry representation	Search technique
1	J. J. K. Dickinson and Knopf 1998 [66]	Geometry	Output maximization	H	Boundary (triangle meshes)	CH
2	J. K. Dickinson and Knopf 2000 [73]	Geometry	Output maximization	H	Boundary (triangle meshes)	CH
3	Teng et al. 2001 [74]	Geometry Problem-dependent	Behavioral objectives	S	Bounding volume (bounding cuboids)	CH
4	Y. S. Wang, Teng, and Shi 2009 [57]	Geometry Problem-dependent	Behavioral objectives	S	Bounding volume (bounding cuboids)	CH
5	Jens Egeblad et al. 2010 [58]	Geometry Problem-dependent	Output maximization	I	Boundary (triangle meshes)	CH
6	Boccia et al. 2011 [42]	Geometry	Output maximization	S	Volumetric (voxel)	CH
7	Fasano 2013 [37]	Geometry Problem-dependent	Output maximization	S	Volumetric (cluster of parallelepipeds)	CH
8	Joung and Noh 2014 [67]	Geometry	Output maximization	S	Boxes	CH
9	Liu et al. 2015 [62]	Geometry	Input minimization	S	Boundary (vertices and facets)	CH
10	Chen et al. 2015 [48]	Geometry	Input minimization Behavioral objectives	H	Volumetric (cluster of pyramidal primitives)	CH
11	Yao et al. 2015 [75]	Geometry	Input minimization	H	Boundary (level-set representation)	CH
12	Attene 2015 [44]	Geometry	Input minimization	H	Volumetric (voxel)	CH
13	Verkhoturov et al. 2016 [43]	Geometry	Input minimization	H	Volumetric (voxel)	CH
14	Ma et al. 2018 [40]	Geometry	Output maximization	H	Boundary (triangle meshes)	CH
15	F. Wang and Hauser 2019 [56]	Geometry Problem-dependent	Output maximization	H	Boundary (triangle meshes)	CH
16	Y. Zhao and Haas 2019 [76]	Geometry	Input minimization	H	Boundary (point cloud)	CH
17	Chekanin 2020 [47]	Geometry	Input minimization	H	Volumetric (clusters of parallelepipeds)	CH

S in geometry complexity: simple

I in geometry complexity: intermediate

H in geometry complexity: high

CH: constructive heuristics

The simplest constructive heuristic is the bottom-left-front (BLF), which is a three-dimensional extension of the two-dimensional bottom-left algorithm. In BLF, objects are sequentially inserted into

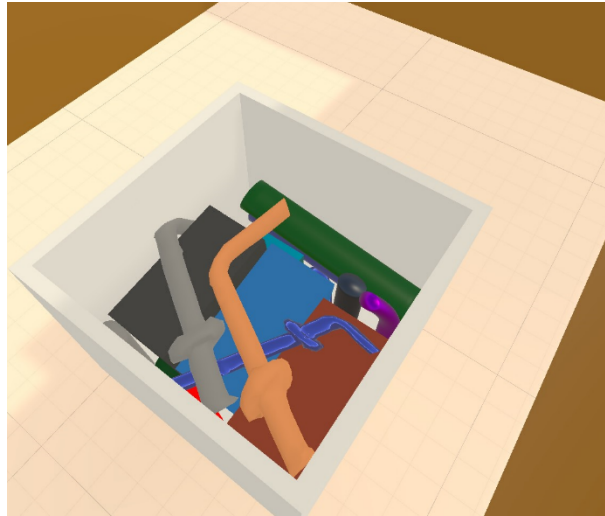


Figure 2-7. A packing configuration generated from BLF

the bottom left front corner of the container's available space. Figure 2-7 demonstrated a packing configuration generated from BLF. Similar to BLF, the majority of constructive heuristics load objects one at a time and rank the location and orientation of each object based on an objective function. For example, Liu et al. propose a constructive heuristic based on the principle of total potential energy minimization. The algorithm finds a placement for the object that reduces the height of the packed object's geometric center by moving the object one by one along the discrete grids inside the container [62]. Rather than packing each object individually, some constructive heuristics group similar objects into subsets before loading them to minimize the number of variables. Egeblad et al. paired sofas as subsets with fragile parts facing each other to facilitate geometric computation and to protect fragile parts during furniture transportation [58].

Apart from the serial constructive heuristics discussed previously, some parallel packing heuristics simultaneously move all objects. Yao et al. begin the random initial layout in a large container with well-separated objects, then shrink the container while removing collisions among objects until the container cannot be shrunk further without causing collisions [75]. Ma et al. expand pre-shrunk objects within corresponding subregions and divide the container space according to the volumes of the objects [40].

One common disadvantage of constructive heuristics is that the packing results are highly dependent on the initialization setup, such as the initial packing sequence of the objects in serial packing heuristics or random placement initialization in parallel packing heuristics. Some preceding

constructive heuristics take steps to increase exposure to potential packing solutions in order to escape the local minimum. For instance, the BLF heuristic is frequently used as a placement heuristic in conjunction with metaheuristics such as the genetic algorithm (GA) in order to optimize the packing sequence and rotations of objects [22,60]. Liu et al. and Zhao et al. used simulated annealing (SA) to optimize the packing sequence in their constructive heuristics [62,76]. Yao et al. iteratively run their parallel parking heuristics with random initializations and choose the final result with the best objective function value [75]. Ma et al. included several heuristic methods, such as swapping and replacing objects, to circumvent the packing results' reliance on initialization [40].

Overall, table 2-1 demonstrates that constructive heuristics approaches can be used to solve 3D irregular C&P problems with a variety of constraint and objective types and objects with varying degrees of geometry complexity. This is primarily because constructive heuristics are intuitive to implement and do not require a great deal of computational effort. As a result, they are frequently quick and capable of producing adequate solutions [77,78]. Constructive heuristics are well-suited for practical problems in civil engineering due to their rapid processing time and their ease of aggregate or integration into a higher-level metaheuristic described in the following section.

2.8.2. Metaheuristics

A metaheuristic is a high-level heuristic that provides guidelines for developing a process capable of escaping local optima and finding good solutions. In contrast to constructive heuristics, metaheuristics are typically problem-independent and can be applied to a wide variety of different problems. Due to the impossibility of conducting an exhaustive search for the optimal solution in many interesting real-world optimization problems, metaheuristics become a critical class of solution approaches. Table 2-2 summarizes publications involving metaheuristic approaches. The most frequently used metaheuristics for solving 3D irregular C&P problems are population-based metaheuristics such as genetic algorithm (GA) and local-search-based metaheuristics such as simulated annealing (SA).

Table 2-2. Publications based on metaheuristics

Number	Publication	Constraint type	Objective type	geometry complexity	Geometry representation	Search technique
1	Szykman and Cagan 1997 [20]	Geometry	Input minimization	S	Bounding volume (bounding cuboids)	M (SA)
2	Campbell, Amon, and Cagan 1997 [79]	Geometry Problem-dependent	Input minimization Behavioral objectives	S	Bounding volume (bounding cuboids)	M (SA)
3	Cagan, Degentesh, and Yin 1998 [45]	Geometry Problem-dependent	Output maximization Behavioral objective	H	Volumetric (Octree)	M (SA)
4	J. K. Dickinson and Knopf 2002 [65]	Geometry	Output maximization	H	Volumetric (voxel)	M (SA)
5	X. Zhang et al. 2002 [21]	Geometry	Input minimization	I	Bounding volume (bounding cuboids)	M (SA)
6	Jang and Rhee 2004 [59]	Geometry Problem-dependent	Output maximization Behavioral objective	S	Cylinders	M (SA)
7	Stefan and Paul 2015 [46]	Geometry	Input minimization	H	Volumetric (Spheretree)	M (SA)
8	Y. Zhao, Rausch, and Haas 2021 [80]	Geometry	Output maximization	H	Boundary (triangle meshes)	M (SA)
9	Lewis et al. 1998 [81]	Geometry	Input minimization	I	Boundary (triangle meshes)	M (GA)
10	J. Hur et al. 2000 [69]	Geometry	Input minimization	S	2D cross-section slice	M (GA)
11	S. M. Hur et al. 2001 [22]	Geometry Problem-dependent	Output maximization	H	Volumetric (voxel)	M (GA)
12	Lewis et al. 2005 [82]	Geometry	Input minimization	I	Boundary (triangle meshes)	M (GA)
13	Yang et al. 2008 [70]	Geometry	Input minimization	S	Bounding volume (bounding cuboids)	M (GA)
14	Gogate and Pande 2008 [60]	Geometry	Input minimization Behavioral objective	H	Volumetric (voxel)	M (GA)
15	Tiwari, Fadel, and Fenyes 2010 [64]	Geometry	Output maximization	H	Volumetric (voxel)	M (GA)
16	Nebel, Richter, and Weicker 2012 [83]	Geometry Problem-dependent	Input minimization	S	Volumetric (clusters of parallelepipeds)	M (GA)
17	Wu et al. 2014 [71]	Geometry	Input minimization Behavioral objective	H	Volumetric (voxel)	M (GA)
18	Fakoor, Ghoreishi, and Sabaghzadeh 2016 [8]	Geometry	Behavioral objectives	S	Bounding volume (bounding cuboids)	M (GA)
19	Araújo et al. 2019 [18]	Geometry	Input minimization	S	Boundary (triangle meshes)	M (GA)
20	J. Zhang, Yao, and Li 2019 [84]	Geometry	Behavioral objectives	H	2D polygon projection	M (GA)

21	Yin and Cagan 2000 [63]	Geometry Problem-dependent	Output maximization Behavioral objectives	H	Volumetric (Octree)	M (PS)
22	Aladahalli, Cagan, and Shimada 2007 [85]	Geometry	Output maximization Input minimization	H	Volumetric (Octree)	M (PS)
23	Aladahalli, Cagan, and Shimada 2007 [86]	Geometry	Output maximization Behavioral objectives	S	Volumetric (Octree)	M (PS)
24	Jens Egeblad, Nielsen, and Odgaard 2007 [5]	Geometry	Input minimization	S	Boundary (triangle meshes)	M (GLS)
25	Jens Egeblad, Nielsen, and Brazil 2009 [68]	Geometry	Input minimization	S	Boundary (triangle meshes)	M (GLS)
26	J. Egeblad 2009 [87]	Geometry Problem-dependent	Behavioral objectives	S	Boundary (triangle meshes)	M (GLS)
27	Vanek et al. 2014 [4]	Geometry	Input minimization Behavioral objectives	H	Boundary (triangle meshes)	M (Tabu search)
28	C. Zhao et al. 2020 [88]	Geometry	Input minimization	S	Volumetric (voxel)	M (Firefly)

S in geometry complexity: simple

I in geometry complexity: intermediate

H in geometry complexity: high

M: metaheuristics

GLS: Guided local search

The SA algorithm is a metaheuristic that uses local search to approximate the global optimum of an optimization problem. It generates a new solution at each step by applying random moves to the current one. If the new solution has a higher objective function value, it is accepted. SA accepts a worse new solution at a certain probability, which gradually decreases to zero, in order to avoid being trapped in local optimization. In the context of 3D irregular C&P problems, a packing solution is a collection of variables (translations and rotations of objects, as described in Section 2.4). Typically, objectives are expressed as objective functions with constraints added as penalized terms. Due to the metaheuristic's versatility, it can be used to consider multiple objectives and constraints concurrently and be easily added to the objective function. SA was introduced by Szykman et al. to maximize the packing efficiency of a three-dimensional component layout problem [20]. The components are represented by blocks or cylinders and are capable of rotation in multiples of 90 degrees around the orthogonal axis. Jang et al. and Cagan et al. propose comparable SA approaches for solving three-dimensional irregular C&P problems with multiple objectives and constraints [45,59].

GA, on the other hand, is a population-based metaheuristic inspired by natural selection. Each packing solution is regarded as a chromosome or genotype. A GA is always initiated by establishing a population of candidate chromosomes. At each generation, only the chromosomes with the highest fitness values are chosen for replication in the following generation via crossover and mutation, resulting in a high-quality solution. GA terminates when a predefined number of generations or time

limit is reached. Ikonen et al. describe one of the earliest implementations of GA, dubbed genetic algorithm for rapid prototyping (GARP). GARP chromosomes contain information about the objects' sequence, orientation, and attachment points between two objects [89]. Hur et al. used GA in conjunction with a BLF algorithm [22]. GA directs the search through modifying packing sequences and object orientations, while BLF generates packing configurations by placing objects in accordance with the information encoded in each chromosome. This ensures that each packing solution is feasible, that no geometry constraints are violated during the search process or in the final solution but limits the exploration of the space of all feasible solutions. Arajo et al. compared the performance of various BLF-based methods, including a brute force algorithm and a GA-based BLF method [18]. According to their research, brute force search is unreasonably slow for 3D irregular packing problems with a large number of objects when compared to GA.

Additionally, researchers have investigated other metaheuristics as search techniques, such as pattern search (PS). As with SA, PS begins with an initial solution and makes random moves to explore the space of all feasible solutions. PS's moves, on the other hand, follow a restrictive pattern, gradually decreasing the step size until it is less than a predefined value. PS was presented by Yin and Cagan [63] and Aladahalli et al. [85,86] for solving 3D irregular C&P problems with a variety of objectives and constraints.

Metaheuristics are high-level heuristics that are not problem-specific; as a result, they are adaptable and simple to implement. As a result, approximately half of the publications reviewed in this thesis used metaheuristic approaches to solve various 3D irregular C&P problems involving a variety of problem types, as illustrated in table 2-2. As a result, metaheuristics can easily be applied to three-dimensional irregular C&P problems in civil engineering. Compared to constructive heuristics, Metaheuristic approaches, on the other hand, are typically more time-consuming than constructive heuristics as they attempt to escape from local optima.

2.8.3. Mathematical Programming

Apart from heuristic approaches, some researchers also attempted to use mathematical programming to solve 3D irregular C&P problems. Leao et al. reviewed the mathematical models for irregular C&P problems in detail [1]. Due to the difficulty of developing mathematical representations for three-dimensional irregular geometries and the multiplicity of objectives, it is still implausible for existing mathematical programming solvers to find an exact solution for three-dimensional irregular C&P

problems. Several methods attempt to decompose 3D irregular C&P problems into a series of mathematical programming subproblems and then use heuristics to better explore the solution space and avoid local optima. Solution approaches, which attempt to combine exact mathematical programming with heuristics, are designated as Matheuristics [1]. To emphasize the mathematical programming aspect of these approaches, they are referred to as mathematical programming in this thesis. Table 2-3 summarizes publications that used mathematical programming.

Table 2-3. Publication based on mathematical programming

Number	Publication	Constraint type	Objective type	geometry complexity	Geometry representation	Search technique
1	Y. G. Stoyan et al. 2005 [49]	Geometry	Input minimization	S	Mathematical (phi-function)	MP
2	Yu Stoyan and Chugay 2009 [90]	Geometry	Input minimization	S	Mathematical (phi-function)	MP
3	Y. Stoyan, Pankratov, and Romanova 2016 [51]	Geometry	Input minimization	S	Mathematical (quasi-phi-function)	MP
4	Stoian et al. 2018 [91]	Geometry	Input minimization	S	Mathematical (quasi-phi-function)	MP
5	T. Romanova et al. 2018 [7]	Geometry	Input minimization	I	Mathematical (quasi-phi-function)	MP
6	Litvinchev, Pankratov, and Romanova 2019 [92]	Geometry	Input minimization	I	Mathematical (quasi-phi-function)	MP
7	Y. G. Stoyan and Chugay 2020 [93]	Geometry	Input minimization	I	Mathematical (quasi-phi-function)	MP
8	Pankratov, Romanova, and Litvinchev 2020 [94]	Geometry	Input minimization	I	Mathematical (quasi-phi-function)	MP
9	Tatiana Romanova et al. 2020 [95]	Geometry Problem-dependent	Behavioral objectives	I	Mathematical (phi-function)	MP
10	A. Chugay, Pankratov, and Romanova 2020 [96]	Geometry	Input minimization	I	Mathematical (quasi-phi-function)	MP
11	A. M. Chugay and Zhuravka 2021 [97]	Geometry	Input minimization	I	Mathematical (quasi-phi-function)	MP

S in geometry complexity: simple

I in geometry complexity: intermediate

MP: mathematical programming

The approach based on the phi-function is the most well-known in this category. As discussed in Section 2.5.1, the phi-function is a mathematical tool that is used to denote the geometric distance between two phi-objects [1]. If two objects are separated, the phi function returns a value greater than zero; if their boundaries touch, the phi function returns a value equal to zero; and if two objects

overlap, the phi function returns a value less than zero. Not only does the phi-function allow for collision detection, but it also enables mathematical modelling of packing problems by mathematically describing non-overlapping and containment constraints. By utilizing the phi- and quasi-phi-functions, it is possible to formulate 3D irregular C&P problems as a non-convex nonlinear programming problem. In 2000, Stoyan and Patsuk used the phi-function to study the two-dimensional lattice packing problem [98]. Their research group has remained the primary users and developers of this methodology since then. Stoyan et al. extended the research on the phi-function to three dimensions by attempting to pack three-dimensional convex polytopes into a parallelepiped [49]. Stoyan et al. introduced the quasi-phi-function in 2016, which is claimed to be significantly simpler than phi-functions for certain types of objects, such as ellipses [51]. While an exact mathematical formulation is a good first step toward achieving optimal global solutions in theory, the current state-of-the-art solver is still incapable of solving 3D irregular C&P problems. To find a good local optimization solution, heuristics are used to break the problem down into smaller subproblems with fewer constraints that can be solved using a nonlinear programming (NLP) solver. Romanova et al. propose using a fast starting point algorithm (FPPA) to obtain a feasible solution with a sufficiently large container, followed by a compact procedure (COMPOLY) that eliminates redundant constraints (e.g., pair of polyhedra with non-overlapping bounding boxes) to generate the subproblems [7]. NLP-solver is used to resolve the resulting NLP subproblems in order to find local optimal solutions.

The phi-function-based mathematical programming method is computationally intensive. Before applying this method, phi-functions for each pair of objects must be constructed. Analytically, constructing phi-functions for non-primary arbitrary shapes can be complicated, as complex shapes are composed of unions and intersections of primary shapes. At the moment, there is no algorithmic method for determining the unions and intersections of primary shapes and generating the phi-function for complex geometries [99]. As a result, this method is currently limited to 3D irregular C&P problems involving identical or weakly heterogeneous object sets with simple to intermediate geometries, such as polyhedra, and is therefore inadequate for most civil engineering applications. Despite these disadvantages, the phi-function is the most advanced mathematical programming method for packing identical, or weakly heterogeneous simple three-dimensional irregular objects.

2.9 Discussion

2.9.1. Trade-offs

As mentioned in Section 2.8, none of the present approaches to solving the 3D irregular C&P problem can find a globally optimal solution in polynomial time. When using or creating solution approaches for civil engineering applications, trade-offs might be made in order to attain good results in a reasonable length of time.

The level of detail (LOD) of geometric representation and the computing time of a solution approach might be traded off. An object can be thought of as a collection of fundamental components such as lines, planes, and surfaces [100]. Variable levels of detail require a different number of fundamental components for the same collection of objects, affecting both the size of the storage and the amount of computational time required. For example, different levels of resolution in the mesh representation imply a varied number of faces and vertices, requiring different amounts of memory and processing time for geometry [101]. That is why numerous geometric simplification approaches reduce the number of faces or vertices to a lower LOD in order to achieve faster graphics performance. Certain businesses, such as 3D printing, require that objects have a high LOD for adequate printing quality, however in many other sectors like construction, a high level of detail might not be required as long as the permitted engineering tolerance is met. Choosing the right LOD for the geometry representation can be advantageous when using or creating solution approaches for specific applications.

Similarly, rotation of an object can alter the computation time. In contrast to the 3D regular C&P problem, where cuboids are normally arranged perpendicular or parallel to the orthogonal axis by default, the default orientations for irregular geometries are obscure. Almost all of the heuristics- and metaheuristics-based techniques discussed in this article consider discrete or near-continuous rotations (continuous within the engineering application tolerance with small enough rotation increments). Often, in order to reduce calculation time, objects are constrained to discrete orientations with a finite number of rotations (e.g., multiples of 90 or 45 degrees) or no rotation at all. In certain scenarios, desired orientations are established for safety or quality reasons; for example, the orientation of sofas is predefined to avoid injuring the delicate portions during furniture transit [58]. Allowing for more flexibility in the rotation of objects in heuristics- and metaheuristics-based algorithms can aid in the exploration of possible packing configurations. However, trade-offs between

the possible performance advantage from increased rotation freedom and the additional computational work required for application purposes must be considered. Continuous rotation of 3D irregular objects has been addressed in mathematic programming techniques and is a possible area of future research.

2.9.2. Research Gaps and Future Research Opportunities

Over the last two decades, researchers have made significant progress in the area of 3D irregular C&P problems. However, there is still considerable space for integrating novel methodologies and refining established approaches.

Manual intervention. Three-dimensional irregular C&P problems are NP-hard, which means they cannot be solved in polynomial time. At the moment, none of the available methodologies is capable of identifying a globally optimal solution. Given that humans are naturally superior to machines at processing geometry data, allowing for human interaction during the packing process can potentially considerably improve packing results. To this end, research needs to be conducted to determine appropriate problem scales and the most effective strategies for incorporating human input or direction into the packing procedure.

Game engines. In civil engineering applications, real-world physics needs to be considered. Game engines such as Unity3D and Unreal, with their integrated physics engines that simulate real-world physics (e.g., gravity and collision), are great platforms for packing planning and simulation, interactive human intervention, and verifying physical constraints (e.g., stability or manipulation feasibility). Effective leveraging of these tools needs to be explored.

Reinforcement learning. With the recent advance, reinforcement learning (RL) has been applied to combinatorial optimization problems like the traveling salesman problem and the knapsack problem [102]. Some researchers have implemented RL to bin packing problems [103]. The development of RL approaches for 3D irregular C&P problems holds enormous promise.

Algorithm for creating phi-functions autonomously. As indicated in Section 2.8.3, generating the phi-function analytically for non-primary arbitrary forms can be difficult due to the fact that complex shapes are built of primary objects formed through unions and intersections. Algorithmic methods for generating phi-functions for complex irregular geometries will bolster approaches based on phi-functions.

Hybrid approaches. Existing search techniques, which are discussed in this article, each have their own set of advantages and limitations. For example, it is still implausible to use mathematical programming to determine the exact solution to 3D irregular C&P problems. Current approaches decompose three-dimensional irregular C&P problems into a succession of mathematical programming subproblems and explore the solution space using heuristics. The combination of several search techniques and the incorporation of novel techniques offers tremendous potential and a plethora of research options.

Apart from the aforementioned research prospects for solution approaches, many pertinent problems have been addressed infrequently or not at all, posing additional challenges and opportunities.

Multiple containers and allocation of objects. 92.8% of the publications reviewed in this research address 3D irregular C&P problems using a single container; nevertheless, many practical 3D irregular C&P problems frequently require several containers, for example, construction waste packing optimization. Efficient allocation algorithms that properly arrange objects into different containers for improved packing results are crucial for future study in 3D irregular C&P problems involving multiple containers.

Online or real-time packing optimization. In each of the 3D irregular C&P problems discussed in this work, the geometry of the objects is specified in advance. Solution techniques seek to optimize certain goals by determining the locations and orientations of objects simultaneously. Situations in which objects come sequentially without prior knowledge of the subsequent objects are referred to as online or real-time C&P problems. In such real-time packing challenges, solution approaches are required to rapidly determine the local best location of each arriving object, e.g., warehousing.

Partitioning and packing problem. A new issue has arisen as a result of the 3D printing industry's growing influence. Large items can be partitioned into printable components and printed concurrently to save time. Similarly, structures must be decomposed and then packed during D&D (as shown in Figure 2-8). Planning and optimizing the structure decomposition process in order to facilitate efficient packing of the resulting components can result in millions of dollars in savings. This is referred to as the partitioning and packing problem or decompose-and-pack. The interaction of decomposition and packing presents both challenges and opportunities for research.

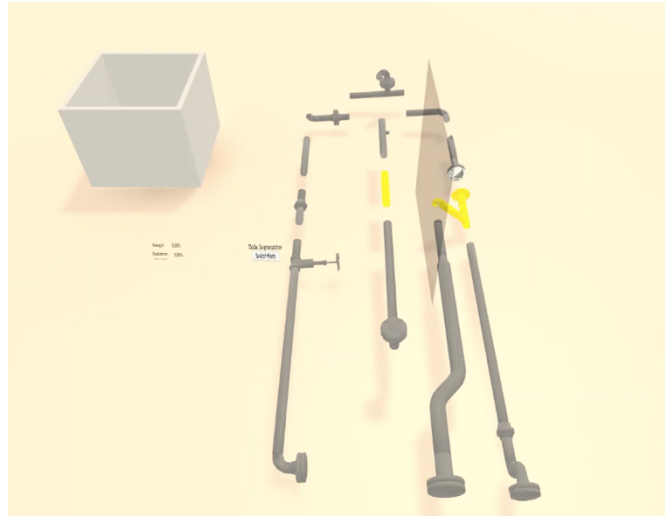


Figure 2-8. Decomposition of a large component

New problem domains. Though mentioned earlier, almost no-one has tackled the dry-stacking problem, yet the demand for dry-stacking might increase, if a good C&P solution could be found. Extending this idea to re-use of construction materials such as windows, doors, and structural elements might result in creative computation or generative architectural design solutions. Urban planning studies could apply similar approaches to study potential for infilling and densification strategies for example.

2.10 Conclusion

The increasing use of technology in construction and civil engineering has created a need for research into 3D irregular C&P problems, which presents a challenge. This study examined fifty-six related publications published between 1995 and 2021 and provided an in-depth examination of the pertinent issues and state-of-the-art solution approaches with a particular emphasis on civil engineering.

First, the following five major constituent components of a 3D irregular C&P problem are identified and summarized: variables, geometry representations, constraints, objectives, and optimization search techniques. For each component, a comprehensive review is provided. Finally, current research on 3D irregular C&P problems is in its infancy and is largely absent from construction applications, implying a wealth of opportunity for addressing novel problem variations, refining existing solution approaches, and incorporating novel techniques such as deep reinforcement learning and game engines.

The following two chapters of this thesis attempt to close some of the previously mentioned research gaps and serve as a reference for the development of novel user-friendly tools, workflows, and methodologies for 3D irregular C&P problems in civil engineering. One gap that motivated the following chapters is that current approaches only consider perfect synthetic watertight models of 3D irregular objects derived from computer-aided design software, which are frequently unavailable in practice or are inconsistent with reality when they do exist. For civil engineering applications, such as optimal packing of NPP waste, there are no perfect as-built synthesized models. Chapter 3 presents a new methodology incorporating data acquisition and packing optimization for optimal packing of as-is 3D irregular objects. Following the proposed methodology, a metaheuristic-based packing optimization algorithm is derived using the commercial software Rhinoceros®. Another gap that motivated this research is the inefficiency of existing fully autonomous methodologies in solving real-world application problems, as the majority of autonomous solution approaches are computationally expensive, time consuming, and ignore fundamental physics such as gravity. In this regard, Chapter 4 discusses the integration of human input and the exploration of game engines and virtual reality techniques in order to discover novel solution approaches.

CHAPTER 3

OPTIMIZING 3D IRREGULAR OBJECT PACKING FROM 3D SCANS USING METAHEURISTICS²

3.1 Summary

For efficient construction-assemblies transportation, volume constrained 3D printing, dry stacking, and facility waste management, a common problem must be solved. It is the practical problem of packing irregular 3D rigid objects into a container with fixed dimensions so that the volume of the final packed objects is minimized. To solve this problem, a methodology is presented that begins with capturing the initial as-is 3D shape data for each object, followed by a metaheuristic-based packing optimization algorithm. This methodology is demonstrated to be effective in two situations with known optimum solutions and in a third situation involving packing of real-life as-is objects. A high-level selection algorithm that is designed to guide the search of possible object subsets, when not all objects can fit into a single predefined container, is discussed as well. Performance is examined for variations, and a preliminary sensitivity analysis is performed. The methodology and its key algorithms are demonstrated to produce effective packing solutions in a mostly automatic manner. Object packing for this class of applications in civil engineering can thus be potentially improved in terms of outcome efficiency and level of planning effort required.

3.2 Introduction

Cutting and packing (C&P) problems have been studied in different disciplines such as mathematics [7,37,55], computer science [68,75], and engineering [64,76,104], because of their broad applications and the substantial potential impact of good solutions. Despite the growing interest and years of research, limited progress has been made on the three-dimensional irregular packing problem; the most challenging C&P problem and one that has become particularly relevant to emerging civil engineering applications such as construction-assemblies transportation, 3D printing, dry stacking, and waste management.

² This chapter is adapted from **Zhao, Y., Rausch, C., & Haas, C. (2021). Optimizing 3D irregular object packing from 3D scans using metaheuristics. *Advanced Engineering Informatics*, 47, 101234. DOI: <https://doi.org/10.1016/j.aei.2020.101234>**

As a working definition of irregular objects, pragmatism prevails in this thesis. Irregular objects are defined as requiring a high degree of geometric granularity and complexity in their digital representation (an analytical function or a limited set of parameter primitives do not provide adequate accuracy in practice for describing irregular objects). For example, an inadequate geometric representation might fail to represent concavity that would yield potential solutions for nesting. By targeting the practical packing problem of arranging irregular objects (with zero deformation) into a box-shaped container with fixed dimensions, such that all objects are enclosed in a minimum volume inside the container and no objects overlap, the scope of this thesis encompasses the applications identified above. Mechanical characteristics, such as equilibrium and stability, are currently out of the scope of this chapter.

A prevalent application of the 3D irregular rigid object packing problem is additive manufacturing, also known as 3D printing; a type of manufacturing method that produces 3D shapes by adding layers. Packing several objects into one build-chamber and simultaneously printing them by batch production can significantly increase manufacturing efficiency and reduce the need for supporting material [3,4]. Another implementation arises in facility decommissioning and disassembling, especially in nuclear waste management, where labor and materials costs are generally higher, and norms and requirements are strict. Decommissioning and disassembling of nuclear power plants produce large amounts of metal and concrete debris with varying levels of contamination. Optimal packing of this waste is a concern for safety, environment and decommissioning costs and has enormous potential in reducing manual packing trial and error (and resulting radiation exposure to humans), the number of containers required, and, eventually, the total overall project cost [76]. Furthermore, the 3D irregular rigid object packing problem can be generally applied in the construction industry, such as assisting in the placement of construction materials on construction sites, improving transport efficiency of building parts or fabricated construction assemblies (e.g., pipe spooling optimization problem presented by Al-Alawi et al. [104]), and reducing the storage space of the construction waste. Esoteric applications such as dry stacking also exist, which will support increased re-use of construction materials and natural stone in a circular economy.

Difficulties of 3D irregular object packing problems come from computational processing limitations and irregular shapes' geometric complexity. Some researchers have proposed approaches for simplifying irregular shapes using limited vertices and plane surfaces such as bounding boxes and polyhedral with vertices less than 30 [19,21,92]. However, these shapes are limited in the accuracy

and precision of their representation of real-world objects, which often exhibit high complexity. Some researchers use more accurate representations, such as meshes or voxels [22,48,60,75]. Nonetheless, their approaches use synthetic idealized watertight models for 3D irregular objects, usually absent in practice or inconsistent with reality if they exist. This thesis utilizes scanned data derived from objects' as-found, as-broken, as-cut, as-designed, or as-built states, defined here using the term "as-is," to ensure veracity and reliability of the geometry representations. The main contribution of the work presented in this thesis is a 3D packing optimization methodology and algorithms for scanned 3D irregular objects. This approach keeps the as-is shape of the irregular objects, which can provide more accurate and potentially denser packing configurations and has numerous potential industrial applications.

The steps followed to develop a solution to the 3D irregular object-packing problem include deriving a methodology for the problem, applying it to as-is objects in a series of experiments and functional demonstrations, and then exploring the performance of the methodology under different situations. As opposed to existing methods that start with simplified shapes or synthetic idealized watertight models, the input geometries of the proposed methodology are highly complex and are derived from objects' as-is state. The proposed methodology is composed of two main steps: (1) data acquisition and (2) packing optimization. The methodology starts with geometric data acquisition by 3D scanning of target objects. Scanned data is then transformed into mesh representations and fed into the packing optimization algorithm. The packing optimization algorithm is based on metaheuristics and allows six degrees of freedom for each rigid object (i.e., rotations and translations about Cartesian axes). Depending on different needs in practical applications, the degrees of freedom and choice of metaheuristics can be customized and adjusted. For demonstration purposes in this thesis, all six degrees of freedom for each object are effectively continuous (very small discrete increments are used) and parallel. The following sections clarify the knowledge gap and summarize the literature on which the research presented in this thesis builds.

3.3 Related Work

3.3.1. Geometry Representation

For the irregular 3D object-packing problem, the handling of object geometry representation is the first obstacle to face. As opposed to regular shapes (e.g., rectangular, cylindrical, prismatic, etc.), irregular forms cannot be defined parametrically using a shape type and a limited set of parameter

primitives that specifies the object. Irregular shapes are often represented either by their enclosing surfaces (e.g., meshes, point clouds, surface normal maps) or a list of spatial cells that the object occupies (e.g., voxels) [41]. Such representations result in more computationally demanding collision detection techniques that are necessary for checking the packing constraints between irregular objects and packing volumes.

Most of the early research in this area represented irregular objects using regular shape primitives like bounding boxes or bounding cylinders to reduce the computational demand in geometry analysis and processing [20,21]. Nonetheless, these approximation approaches were proven ultimately to be inefficient [22], and their accuracy is not adequate for the applications on which this thesis is focused.

Researchers then progressed from the bounding shape representation scheme to use of representations that more accurately fit irregular 3D objects such as mesh, voxel, and level set representations [4,45,60,75]. The escalation in the number of vertices or the number of spatial cells used in more accurate geometry representation leads to increased computational complexity. This has largely been accepted as a reasonable tradeoff, since the computational processing power of computers has significantly evolved in recent years, thus easing the imperative on compact data structures. However, these approaches work on perfect synthetic watertight models of 3D irregular objects, usually absent in practice or inconsistent with reality if they exist. The critical difference between the proposed methodology and the previous work is that, instead of using simplified shapes or synthetic idealized watertight models, this work utilizes scanned data derived from as-is objects, which ensures accuracy and applicability for numerous industrial applications. By applying the proposed methodology, quick crude scans are sufficient to generate good packing layouts for many as-is objects.

3.3.2. Mathematical Modeling-based Approaches

Researchers have previously developed mathematical modeling-based approaches for object packing. The phi-function is the most popular mathematical tool. It is used to provide analytic descriptions of spacial relations for 2D polygons or 3D polyhedra [7]. Stoyan et al. proposed a mathematical programming approach based on the phi-function to pack 3D polytopes into a box of minimal height [105]. Romanova used quasi phi-functions, a simpler version of the phi-function, for packing non-convex polyhedra with into a cuboid of minimal size. They refer to the problem as the polyhedron packing problem [7]. However, their mathematical modeling approach is limited to polyhedra having

less than a few dozen vertices. When surveying these and other existing methods for mathematical modeling-based approaches, it becomes clear that they function well when object complexity (i.e., the number of vertices) is small and therefore accuracy for typical irregular objects is poor. As such, they suffer when processing complex or large numbers of vertices, since they do not scale well.

Essentially, this is because they link collision detection (constraint checking) and solution searching in a way that does not allow for more complex representations without overwhelming the computational requirements. Thus, for the irregular objects within the scope of the problem addressed in this thesis, the existing mathematical approaches currently are unrealistic and inapplicable.

3.3.3. Heuristics

The 3D irregular C&P problem is known to be Np-hard. In other words, the expected time to find an optimal solution is likely to increase exponentially as a function of the number of inputs [18]. It should also be noted that none of the existing algorithms for the 3D irregular C&P problem can find a globally optimal solution in polynomial time. Heuristic-based algorithms have the advantage of being able to find close-to-optimal solutions in a realistic amount of time [19]. For realistic solutions, the class of heuristic methods, in particular metaheuristics, is the most generally applied class of algorithms for solving irregular 3D object packing problems in practice [27,55]. This is because they simplify the problem complexity by dissociating the collision detection (constraint checking) work (computational load) from the search mechanism's work and allow easy integration of different constraints as well as objectives. For example, Ikonen et al. propose a genetic algorithm (GA) for packing three-dimensional non-convex objects having cavities and holes. This approach utilizes digital representations (voxels) of the actual object geometry. The "attached points" between objects are defined manually by the human operator. This limits the collision detection computational requirement. Each chromosome is a list of packing sequences, orientation, and attached points. A GA guides the search in a direction that optimizes the value of the fitness function [89]. Gogate et al. propose another GA-based algorithm using voxel representation for intelligent layout planning for rapid prototyping with a fitness function comprising other constraints including part build-height, staircase effect, volume, and area-of-contact of support structures. They first generate a list of the possible orientation of objects rating by item printing quality. Then a GA is applied to achieve an optimal packing layout by changing the sequence and orientation [60]. Other metaheuristics such as simulated-annealing, Tabu-search have also been adopted to deal with the irregular 3D object-packing problem. The common process of adopting metaheuristics for the irregular 3D problem is to: (1)

choose a placement heuristic (e.g., bottom-left algorithm) to generate feasible packing configurations with no collisions between objects and between objects and the container (i.e., to minimize the constraint checking computational load); (2) use metaheuristics to guide the search through different packing sequences and objects' orientations. The benefit of this approach is that packing results are feasible physically, however the disadvantage is that only a small portion of the solution space is explored by working through only feasible solutions. The feasible solution, in mathematical optimization, is the set of variables that satisfy all of the optimization problem's constraints. In the context of this thesis, the feasible solution refers to the packing configuration that satisfies the following conditions: (1) all objects are entirely inside the container, and (2) objects do not collide with each other.

Apart from metaheuristics, some researchers have also incrementally and specifically developed constructive heuristics from scratch. The most popular one is the bottom-left-front (BLF) heuristic, which is an extended version of the bottom-left heuristic in the 2D bin packing problem. In BLF, objects are placed in the furthest feasible bottom-left-front position. Wu et al. [71] developed a modified BLF heuristic combined with a GA similar to [60] for a 3D packing problem in additive manufacturing with multiple objectives (e.g., build-height, surface roughness, and support volume). Liu et al. suggest a constructive heuristic algorithm for a 3D irregular C&P problem by following the tendency of lowering the objects' positions [62]. Zhao et al. propose another constructive heuristic with the objective of maximizing the contact area between objects using point cloud representation [76]. However, the two latter heuristics both need to hybridize with metaheuristics like simulated annealing (SA) to improve the packing quality further. Ma et al. [40] propose a constructive heuristic based on hybrid optimization. Objects are randomly selected, and a scaling factor is decreased below 1 (i.e., making the objects shrink). The algorithm then divides the container into separate regions according to the objects' volumes and allows objects to grow individually to their original size. A sequence of operations (object swapping, enlarging, and replacing) are used to escape from strong dependence on placement initialization. The limitation of the method is that the algorithm performs only local optimization, which is a common limitation for constructive heuristics for the irregular 3D packing problem.

In this thesis, a metaheuristic-based packing optimization algorithm is used to achieve quasi-optimal solutions in a realistic amount of time. The proposed packing algorithm uses a fitness function that separates objectives and constraints to encourage the search towards a feasible region. It

also allows for objects being placed outside the container and colliding with each other during the process to promote a broader exploration of the solution space.

3.3.4. Partitioning and Packing Problem

With the burgeoning impact of the 3D printing industry, a new related problem has emerged. Not limited to only packing several objects into one build chamber and print by batch, researchers have started to think about how to decompose a large object into printable pieces to save printing and assembly time. This problem is referred to as the decompose-and-pack problem or partitioning and packing problem. The decompose-and-pack problem not only requires optimizing the packing of pieces but also seeks to decompose 3D objects into rigid pieces which can then be efficiently packed [48]. The coupling of decomposition and packing makes the solution search even harder. Vanek et al. [4] propose an algorithm that converts the 3D model into a shell, which is then divided into segments. The packing part follows the ensuing process. A placement heuristic that minimizes the waste parts between segments is used to build packing configurations. Tabu-search is then used to optimize the packing sequence. Yao et al. [75] propose an iterative process between decomposition and packing to find the partition with high qualities that produces minimum packing volume. The decompose-and-pack problem is a variation of irregular 3D packing problem; however, it is out of the scope of this thesis.

3.4 Proposed Methodology

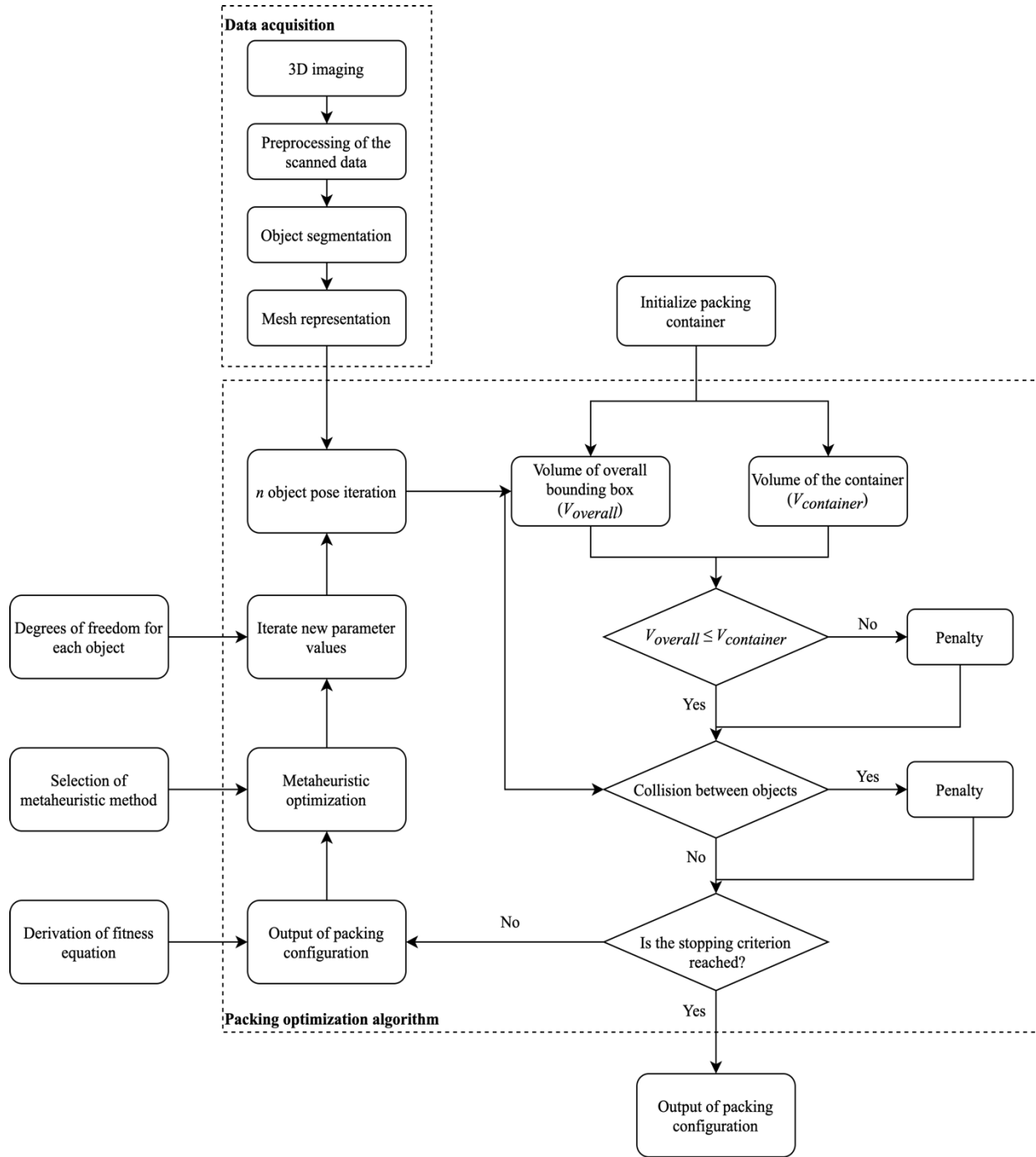


Figure 3-1. Overall methodology for metaheuristic-based optimal packing of irregular objects into a container

The overall proposed methodology for implementing metaheuristic-based optimal packing of objects into a container is outlined in Figure 3-1 and discussed in the following sections. This methodology performs iterative packing optimization on a set of objects in order to minimize a fitness function that drives the convergence process using metaheuristics.

3.4.1. 3D Imaging

The first step involves capturing the initial as-is data for each object to be packed into the specified container. A range of 3D imaging methods can be employed for this step including 3D laser scanners, range cameras, photogrammetry, and structured light scanners. In general, the choice of 3D imaging method depends on the desired data resolution, accuracy and distance between measurement device and objects (i.e., scale of the scene). Review of industrial and construction object centric data acquisition reveals an initial preference towards phase-based terrestrial scanners. First, terrestrial scanners produce dense datasets (millions of data points for each scan) and can facilitate an object capture scale of 0.1m^3 to 20m^3 (i.e., most objects that need to be packed into a container). Since most objects to be packed into a container can be measured at short ranges (1m to 50m), phase-based terrestrial scanners are more accurate than time-of-flight terrestrial scanners which are used for capturing objects kilometers away [106]. Irregular industrial objects do however create challenges for terrestrial scanners in terms of their surface and geometric attributes (e.g., low albedos, high reflectance properties, and low incidence angles). For these reasons, multi-sensor applications are robust for industrial object as-is data capture, as shown by Hullo et al. [107]. While laser scanners struggle to produce reliable data for sharp angles, this is where image-produced data (i.e., photogrammetry) is superior [108]. For solely capturing close range objects (less than 1m), metrology devices such as portable coordinate measuring machines (CMMs) with scanning devices can be employed, which are capable of producing dataset accuracies of less than 0.1mm, and generally are suitable for capturing sharp angle geometry [109].

3.4.2. Irregular Data Processing

The second stage in the methodology involves processing the raw as-is data in order to obtain a mesh representation format. The purpose of producing a mesh format is to facilitate efficient object collision detection within the object packing optimization process. To do this, the scanned data requires preprocessing to remove outliers and noise. Then, objects can be segmented, and saved as a mesh file.

The initial as-is geometry data for irregular-shaped objects is obtained in this study utilizing a structured-light scanner called the Occipital Structure Sensor [110]. The Occipital Structure Sensor generates an .obj file of the scanned scene immediately (Figure 3-2a), which can be imported into MeshLab® [111] for pre-processing to remove outliers and noise before being segmented and stored as mesh files (.obj) for each object shown in Figure 3-2b. The sensor's precision is within the engineering application tolerance of a magnitude of millimeters.

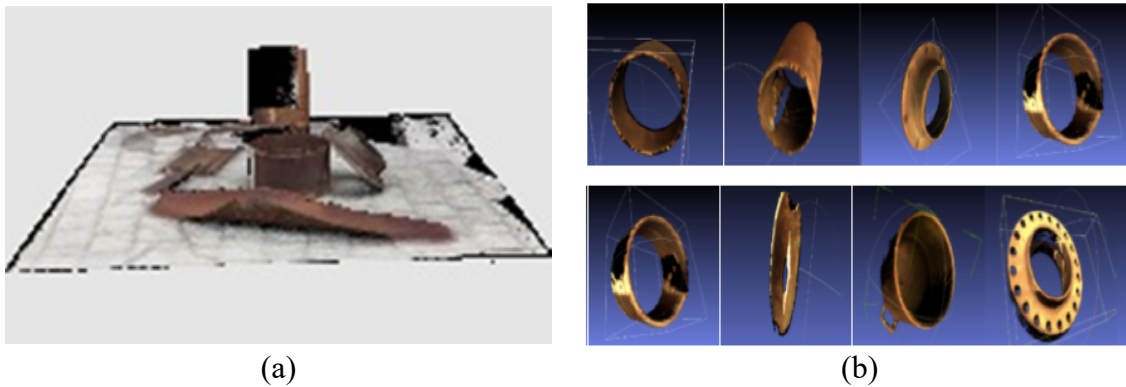


Figure 3-2. (a) Scanned data in .obj file. (b) Mesh files of the objects.

If point cloud data is captured via scanning using a 3D laser scanner, it can be converted to meshes using a variety of algorithms described in [112] that conduct normal and Poisson surface reconstruction [113]. However, in practice, reconstructing surfaces from point clouds faces a variety of challenges, including nonuniform point sampling, noisy normals due to scan inaccuracy and misregistration, and some surface regions devoid of data due to scanning accessibility constraints [114]. As illustrated in Figure 3-3, the method described in [112] may result in an inaccurate surface reconstruction. For simple shaped objects, the inaccurate mesh surface can be manually repaired in MeshLab®. However, manual adjustment becomes inconvenient and imprecise when dealing with complicated items. Automated and precise reconstruction methods for converting point clouds to meshes are an active area of research that may eventually close this gap.

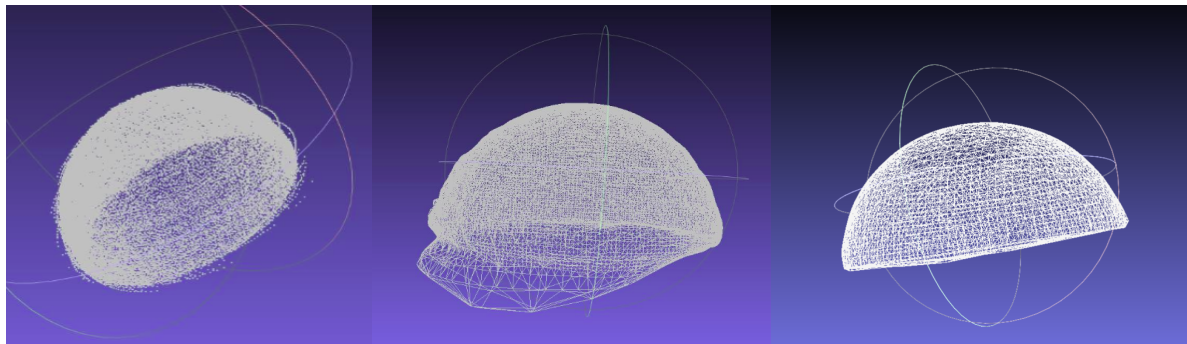


Figure 3-3. Converting the point cloud of a dome-shaped object into a mesh. (a) Point cloud of the dome. (b) Conversion of the point cloud to a mesh. (c) Mesh after manual correction.

3.4.3. Packing Optimization Process

Once objects are represented as meshes, the object packing optimization algorithm can begin. There are four key inputs to this algorithm: (1) initialization of the packing container, (2) specifying the degrees of freedom for object transformation, (3) derivation of the fitness equation, and (4) selection of the metaheuristic optimization method.

Initializing the packing container involves specifying the geometric parameters, and in the context of this thesis, the container is assumed to be a rectangular prism (i.e., shape parameters are length, width, and height). This thesis considers rigid pose transformation parameters that are consistent with rigid objects, which include up to six degrees of freedom (DoF): principal translations in x, y, z and principal rotations about the x, y, z axes respectively. Sometimes, a seventh DoF can be considered for scaling, however it is assumed that the objects and container dimensions are in the same scale in this thesis.

3.4.4. Initialization

The initial placement of the objects is randomly generated in close proximity to the container. The six degrees of freedom for each object are denoted as $(v_x, v_y, v_z, \theta_x, \theta_y, \theta_z)$, where $v = (v_x, v_y, v_z)$ is the vector of translation and $\theta = (\theta_x, \theta_y, \theta_z)$ is the object's rotation about the $x, y,$ and z axes respectively. $L, W,$ and H are the length, width, and height of the container, respectively. The translation and rotation of the container are also considered. The six degrees of freedom of the

container are denoted as $(v_x, v_y, v_z, \theta_x, \theta_y, \theta_z)$, where $v_c = (v_x, v_y, v_z)$ is the vector of translation and $\theta_c = (\theta_x, \theta_y, \theta_z)$ is the container's rotation about the x, y, and z axes respectively.

Object translations are limited within a specific range from the container to avoid searching over meaningless packing layouts. The translation range for each object is defined by the combination of its bounding box and of the container (Figure 3-4). Once a container is defined, an axis-aligned bounding boxes (AABB) is generated for each object. The AABB of each object is brought into contact with the container and translated around the container edges. The translation range is limited by a parallelepiped since the container is a rectangular prism. The translation range is calculated once for initialization.

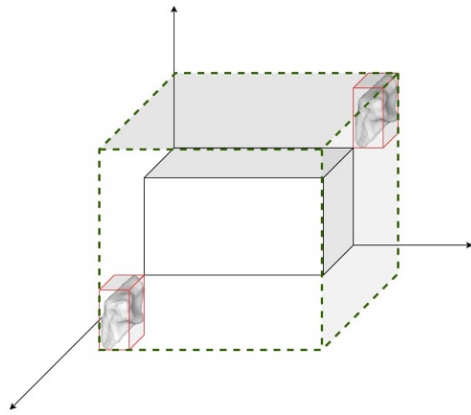


Figure 3-4. Translation range of an object with respect to the container

Within the translation range, an object can move continuously, and collisions are detected using mesh representation of the object. The objects are allowed multi-axis, 360-degree rotation. Increments of rotation and translation can be changed according to problem. With very small discrete increments, the move of an object can be considered continuous within engineering application tolerance. Discrete increment of 0.01 m and 0.1° are used for translation and rotation respectively in the experiments later in this chapter.

3.4.5. Fitness Function

A general optimization problem can be expressed as follows:

$$\text{minimize } f_0(\vec{x}) \tag{3-1}$$

$$s.t. g_i(\vec{x}) \leq 0, i = 1, \dots, m$$

$$h_j(\vec{x}) = 0, j = 1, \dots, p$$

$$x_k^l \leq x_k \leq x_k^u, k = 1, \dots, n$$

where $\vec{x} \in \mathbb{R}^n$ is the optimization vector or a set of decision variables. $f_0: \mathbb{R}^n \rightarrow \mathbb{R}$ is the fitness or objective function of the problem. g_i are inequality constraint functions. h_j are equality constraint functions. $x_k^l \leq x_k \leq x_k^u$ is the bound constraint, the k th variable range from x_k^l to x_k^u . Decision variables, objective, constraints (equality constraint, inequality constraint, and bound constraints for decision variables) are important optimization features. The optimum optimal solution has the smallest value of objective function among all vectors that satisfy the constraints. More information on optimization can be found in [115,116].

Metaheuristics such as GA and SA are developed for the unconstrained and bound-constrained type of optimization problems. The most popular constraint handling method in metaheuristics is adding the constraints as penalty terms in the fitness function [61]. As a result, the constrained optimization problem is converted into an unconstrained optimization problem. Solutions outside the feasible region are penalized. The resulting fitness function is:

$$\text{minimize } f(\vec{x}) = f_o(\vec{x}) + \sum_{i=1}^m a_i \max [0, g_i(\vec{x})] + \sum_{j=1}^p b_j |h_j(\vec{x})| \quad (3-2)$$

where $f(\vec{x})$ is the new fitness function, a_i and b_j are weights assigned to each penalty term, also called the penalty factor.

Compared to other constraint handling techniques such as repair operators, penalty-based constraint handling has the advantage of more straightforward implementation and problem-independence [117]. Deb et al [118] devise a fitness function that separates objectives and constraints:

$$\text{minimize } f(\vec{x}) = \begin{cases} f_o(\vec{x}) & \text{if feasible} \\ f_{worst} + \sum_{i=1}^{m+p} k_i(\vec{x}) & \text{otherwise} \end{cases} \quad (3-3)$$

where f_{worst} is the objective function value of the worst feasible solution. Equality constraints $h(\vec{x})$ are converted into inequality constraints $g(\vec{x})$ by subtracting a small positive value. $k(\vec{x})$ consists of both normalized equality constraints and inequality constraints. Such a fitness function

ensures that: (1) any feasible solution is superior to any infeasible solution, (2) a feasible solution with a better objective value is superior between two feasible solutions, and (3) an infeasible solution with fewer total constraint violations is superior between two infeasible solutions. Later in this thesis, a similar separate function is used to build up a fitness function that encourages the search towards a feasible region.

When only the geometry criterion is considered, the fitness function can simply be a single objective, such as minimizing the final height of the packing or minimizing the volume of the packing configuration. In additive manufacturing, criteria such as staircase error, the amount of the support structure, etc. need to be considered for a good packing layout. In such a case of multi-objective optimization of the 3D packing problem, the primary approach is to aggregate all objectives into a decomposed fitness function such as a weighted sum [71]. In the context of this chapter, only geometric constraints are considered. Other constraints, such as stability, manipulation feasibility, and capacity constraints, are reviewed and addressed in Chapter 4 using an interactive virtual reality packing platform.

Given a container denoted by C , a set of n irregular objects denoted by O (the i th object is denoted by $O_i, i = 1, \dots, n$), and a space range of possible placements S , the goal is to minimize the overall packing volume under the premise that: (1) all objects are entirely inside the container, and (2) objects do not collide with each other. As discussed in section 2.3, by using $O_i \oplus v_i$ to define object O_i is translated by v_i , and $O_i(\theta_i)$ to define the object is rotated by θ_i , the above constraints can be represented by:

$$O_i(\theta_i) \oplus v_i \subseteq C \quad i = 1, \dots, n \quad (3-4)$$

$$(O_i(\theta_i) \oplus v_i) \cap (O_j(\theta_j) \oplus v_j) = \emptyset \quad i = 1, \dots, n, j = 1, \dots, n, i \neq j \quad (3-5)$$

Any collisions and violations of the enclosure need to be avoided and are thus taken into account using penalties. Similar to equation (3-3), the constraints are each added as a penalty term and are separated from the objective in the fitness function. The following equation describes the fitness function:

$$f = \begin{cases} V_{packing} & \text{if } (O_i(\theta_i) \oplus v_i \subseteq C) \cap ((O_i(\theta_i) \oplus v_i) \cap (O_j(\theta_j) \oplus v_j) = \emptyset) \quad \forall O_i, O_j \in O, i \neq j \\ w_a(A - 1) + w_o V_{overlap} + V_{container} & \text{otherwise} \end{cases} \quad (3-6)$$

Where $V_{packing}$ is the volume of packed objects' AABB and $V_{container}$ is the volume of the container (this term corresponds to the term f_{worst} in equation (3-3)). The objective function value of the worse feasible solution is the volume of the container. $V_{overlap}$ is the overlapping volume of two colliding objects' AABB. $A = (V_{overall}) / (V_{container})$, where $V_{overall}$ is the AABB volume of the set including objects and container. A is a measurement that quantifies how well objects are enclosed inside the container. The minimum value of A is 1, which indicates objects are entirely inside the container. Its value increases as the objects exceed the container and move outward.

w_a and w_o are weights assigned to $(A - 1)$ and $V_{overlap}$ terms, respectively. These weights are empirical values and are variable based on the packing problem being considered. When setting different weights to the two parameters, the most important consideration is to set the ratio between w_a and w_o to such a value which ensures neither constraint has excessive dominance over the other. The problem of defining the values of the two weights in practice is thus reduced to finding a balanced ratio between w_a and w_o . Theoretically, a ratio α between w_a and w_o can be defined so that only one parameter must be set. In this case however, w_a and w_o are preserved to keep the fitness function explicit and intuitive in terms of weighted constraints. The ratio α is an empirical value, which can vary for different packing problems. Users can define α from test-running the algorithm for a specific situation. If, during the test run, the order of magnitude of $(A - 1)$ is 0.01 and the order of magnitude of $V_{overlap}$ is 0.1, then α can be set to 10 (e.g., $w_a=10$, $w_o=1$) to ensure neither constraint has excessive dominance over the other.

For feasible solutions, where all objects are entirely inside the container and there is no collision between objects, the goal of the fitness function is to minimize the bounding volume of the packed objects. Otherwise, when the geometry constraints are violated, penalties are added to the fitness function. Infeasible solutions are compared in terms of lower total constraint violations. The fitness function is built in a way such that overlaps-between-objects are allowed (yet penalized) during the search process, which contributes to a broader exploration through the solution space. The fitness function guides the search in a direction that decreases the overlap between objects.

As per the discussion above, the mathematical formulation of 3D irregular C&P problem can be defined as follows:

$$\begin{aligned}
& \text{minimize } f(v_i, \theta_i) \\
& \text{s.t. } O_i(\theta_i) \oplus v_i \subseteq S \\
& i = 1, \dots, n
\end{aligned} \tag{3-7}$$

3.4.6. Metaheuristics

Once the fitness function is derived and the objects' degrees of freedom are defined, the metaheuristic approach, as an efficient search strategy, is used to solve the optimization problem. The most applicable metaheuristics for performing this type of optimization are GA and SA [27,29]. From a series of preliminary experiments using the described fitness function, SA always converged to a better packing layout than GA for our specific packing optimization algorithm. Thus, SA is employed as the default metaheuristic solver.

In the search process, SA starts with one state and updates the state through random moves. In the context of this thesis, the variables are the translation and rotation $(v_x, v_y, v_z, \theta_x, \theta_y, \theta_z)$, of each object and the movement $(v_{xc}, v_{yc}, v_{zc}, \theta_{xc}, \theta_{yc}, \theta_{zc})$, of the container. Each state is a list that aggregates these variables. A move is an operation taken to create a new state by translation and rotation of objects and the container. Within the algorithm, one initial (arbitrary) state is chosen and evaluated by the fitness function. Random moves are taken to create new states. For each new state, if the fitness value is better than the current state, we accept the new state as the current state; if the new state is worse than the current one, the new state may still be accepted with a certain probability defined by the function $P_{accept} = e^{-\Delta E/T}$, which is an analogy with the annealing process of metal. ΔE is the change in the objective function, and T is the current temperature. A high temperature is given initially, the probability of accepting worse solutions is high; as the algorithm progresses, the temperature decreases, so is the acceptance possibility. By accepting worse solutions, SA can avoid being trapped in local optimization. The high acceptance probability at the start allows for a broader exploration of the solution. Over time, lowering acceptance probability makes the algorithm focus on developing good moves and converging to a suitable solution.

The search process is terminated when the stopping criterion is reached. The stopping criterion can be a time limit, an absolute number of generations, a specific pre-defined value the fitness function needs to reach, or a situation where there has been no improvement in the fitness function. Time limits are designated as the stopping condition in this chapter.

3.5 Experiments with Archetypical Situations

In this section, three archetypical situations of varying degrees of computational complexity are introduced with the purpose of testing the performance of the packing optimization algorithm. The selection algorithm for a situation where not all objects can be packed inside the container is discussed at the end of this section. The algorithm is implemented in Rhinoceros® using Grasshopper, a visual programming language and environment. The details of the algorithm can be found in appendix A. Galapagos [119], an integrated solver within Grasshopper, provides the SA solver, while the Grasshopper component (function) “Collision Many | Many” is implemented to test collisions between meshes of the objects. “Collision Many | Many” returns a Boolean list for collision event instances between objects and a list for indices of which objects have these collision events. The case studies are conducted on a computer with an Intel i5-6300 CPU and 8GB of RAM.

3.5.1. Situation 1

To validate the performance of the proposed methodology, an archetypical situation comprising simple parametric geometry is conducted. A $1 \times 1 \times 1 m^3$ cube is divided into $2 \times 2 \times 2$ small cubes of size $0.5 \times 0.5 \times 0.5 m^3$. The small cubes need to be packed into a container of size $1 \times 1 \times 1 m^3$. For this case study, the optimum solution is known.

The packing result after 20 minutes is shown in Figure 3-5. The result shows that all cubes are packed tightly inside the container. No constraint is violated. The final volume of the packed small cubes is $1 \times 1 \times 1 = 1 m^3$, which is exactly the optimum solution. This functional demonstration provides some limited confirmation of the methodology’s potential performance in obtaining a global optimum for simple shaped objects. It does not prove that the packing optimization algorithm will always find a good solution, which is a drawback of almost any metaheuristic-based optimization algorithms and optimization algorithms for irregular 3D object packing. However, it will almost always find a feasible solution, by design. And, though a human could derive this solution in this situation much more quickly, this solution is automatically derived including start and end locations for each object. Thus, as limited as this simple demonstration situation is, its solution provides the basis for automated manipulation and path planning in the future, which in and of itself has value.

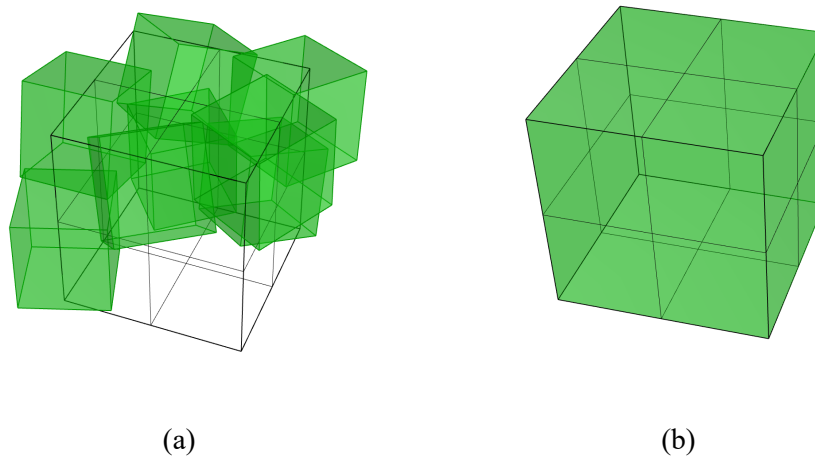


Figure 3-5. Packing result of $2 \times 2 \times 2$ small cubes inside a cube container. (a) Initial placement. (b) Packing result after 20 minutes

3.5.2. Situation 2:

Next, a 3D irregular C&P problem, with a known global optimum, is also created. The instances are created by randomly breaking a rectangular Styrofoam block ($300 \times 200 \times 100 \text{ mm}^3$) into four pieces. The problem is to pack the resulting four pieces into a container, which is slightly bigger than the original rectangular block. The ideal optimum packing layout would be to restore the packing layout, as shown in Figure 3-6, and the optimal volume of the packed pieces is $300 \times 200 \times 100 \text{ mm}^3$. The geometry information of the four pieces is obtained by scanning, followed by pre-processing to delete the noise points from the scan and generate a mesh representation for each piece. The mesh files are then imported into Rhinoceros® and are ready for the packing optimization algorithm. The container is set to be $310 \times 210 \times 110 \text{ mm}^3$. The reason why the container is set to be slightly bigger than the original block is that when scanning the objects, small deviations will occur (there may be some small cavities in the real object that are filled in), which means the digital objects cannot fit perfectly in the original block volume.

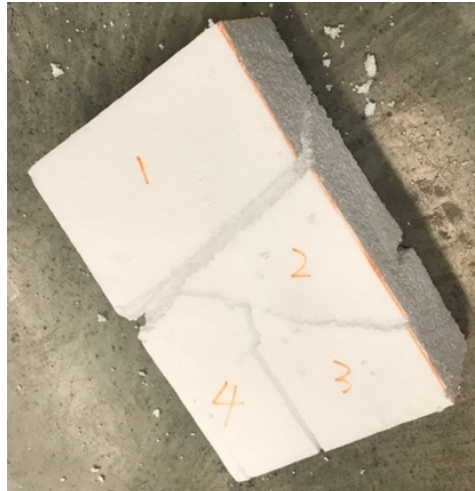


Figure 3-6. Optimum packing of 4 foam pieces

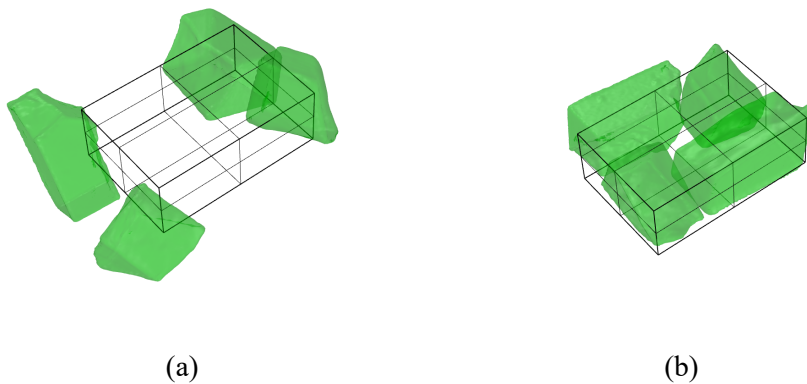


Figure 3-7. Packing result of 4 foam pieces into a container slightly bigger than the original rectangular block. (a) Random initial placement. (b) Packing result after 45 minutes

The packing optimization algorithm is instantiated by randomly placing the four pieces in close proximity to the container. The packing result after 45 minutes is shown in Figure 3-7. The result is not ideal. Even though the objects are tightly packed and there is no collision between objects, the four pieces are not enclosed inside the specified container. The final volume of the packed pieces is $327 \times 271 \times 149 \text{ mm}^3$. The packing optimization algorithm becomes inefficient when the problem's feasible solution is hard to find.

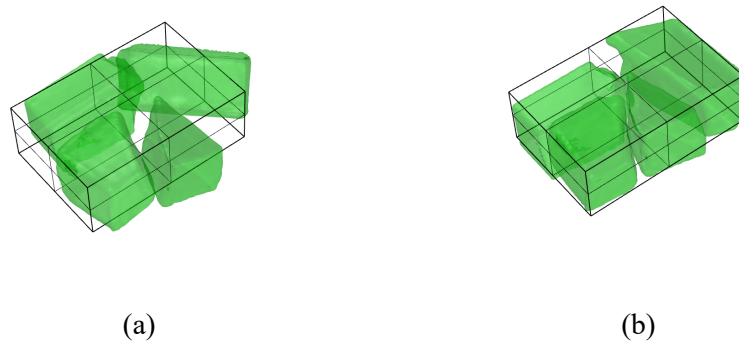


Figure 3-8. Packing result of 4 foam pieces into a container slightly bigger than the original rectangular block. (a) Initial placement that is close to feasible solutions. (b) Packing result after 20 minutes

Then, the four pieces are intentionally placed in an initial placement that is close to a feasible solution. The packing result derived from the packing optimization algorithm after 20 minutes is shown in Figure 3-8. This time, the final volume of the packed pieces is $337 \times 220 \times 144 \text{ mm}^3$. There is no violation of collision constraints however, the four pieces are still outside of the container. This result does however show the tendency of converging towards the optimum solution. The key observation from this study is that: (1) the situation reveals the packing optimization algorithm's inefficiency in facing an optimization problem for which the feasible solution is hard to find, because it is so tightly constrained, and (2) the inefficiency of the packing optimization algorithm can be mitigated by choosing a better initial placement.

Another finding is that this study provides a benchmark instance with high geometric complexity and a known global optimum solution for the 3D irregular C&P problem. Instances with known global optimum solutions make it easier to evaluate the overall performance of certain algorithms. As mentioned in [3], the geometric complexity at the real-life level is lacking in existing benchmarks for 3D irregular packing. The four pieces derived from the broken Styrofoam can fill the gap reported in the literature and well represent real-life geometries. For reference, the mesh files of the four pieces can be found at <https://github.com/ouioui79/Metaheuristic-based-optimization-for-non-parametric-3D-object-packing>.

3.5.3. Situation 3:

For the third archetypical situation, a set of real-life as-is objects is generated. Eight objects are scanned using a commercial 3D structured light scanner. These objects include broken Styrofoam,

concrete debris, and scrap metal pieces. Pre-processing is done in MeshLab to remove redundant points from the scans (noise, scene clutter, etc.).

The objects are to be packed inside a box container of size $0.65 \times 0.55 \times 0.65$ m³. Figure 3-9 shows the result of the packing optimization algorithm after 45 minutes. The result exhibits no constraint violations. From Figure 3-9, it can be noted that some smaller objects are nested inside the hollow parts of larger objects, which can generally increase density of packing inside the container.

The experiments with archetypical situations show that the proposed methodology provides full guidance, from data acquisition to final packing configuration, for 3D packing of irregular objects. The methodology not only suits simple geometry like boxes, but more importantly, it works for complex as-is objects.

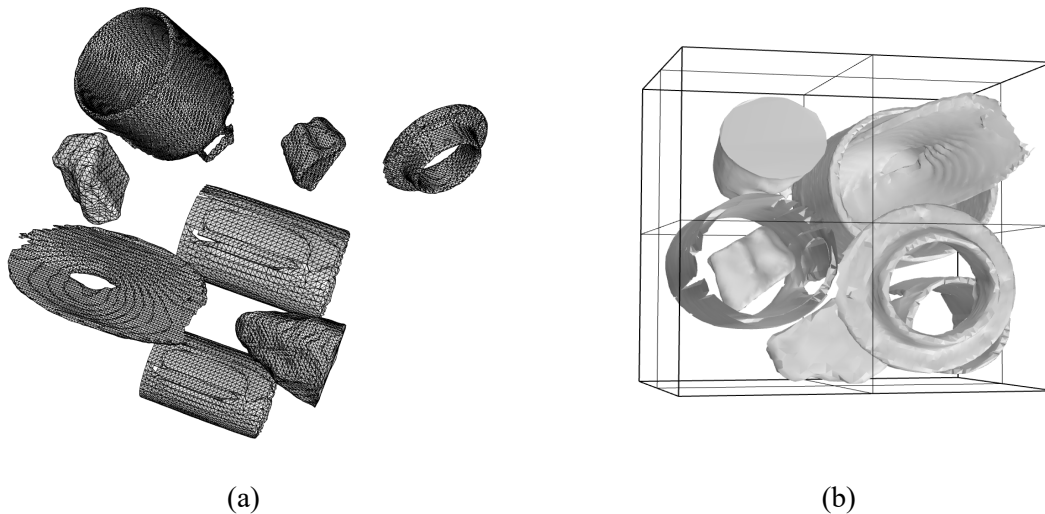


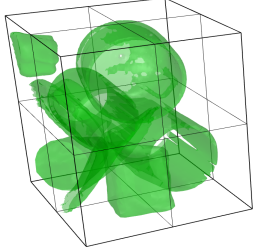
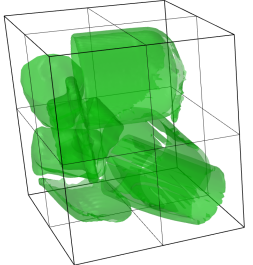
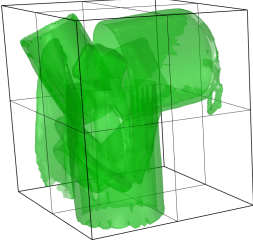
Figure 3-9. Packing result of real-life as-is objects into a rectangular container. (a) 8 as-is objects (b) Packing result after 45 minutes

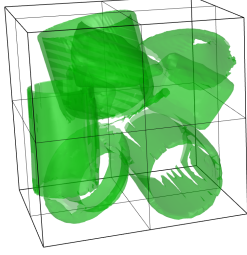
3.5.4. Sensitivity Analysis

Due to the stochastic nature of the packing optimization algorithm, variances in the output packing configurations exist. In order to test the robustness of the packing optimization algorithm, situation 3 is repeated five times, with randomly generated starting positions. The results are summarized in Table 3-1. The test results illustrate variance in final packing configurations, which is mainly the result of different starting positions and stochastic characteristics of the SA algorithm. Two out of five (40%) tests yield feasible solutions, indicating the likelihood of the proposed packing

optimization algorithm to still find packing configurations of good quality despite the variances described. Of the remaining three tests, with no feasible solution found, the amount of constraint violation is quite low, meaning the resulting packing layouts can be easily adjusted by human intervention to become feasible solutions if needed. Furthermore, since the stopping criteria is based on a predefined computational time, there is a chance that feasible solutions could be generated at longer run times. One method to remedy the uncertainty of the proposed packing optimization algorithm is to make multiple attempts and select the best solution [65]. In this case, the solution generated by starting position 2 could be easily and automatically selected as the best solution.

Table 3-1: Packing results of the 8 objects in situation 3 with 5 random starting positions

Starting position	Packing configuration	Feasible solution	Outside container %	Overlapping volume	Fitness value	Time (mins)
1		✘	1.313%	0	0.245506	20
2		✔	0	0	0.199999	20
3		✘	0.2705%	0.002771	0.26279	20

4		✗	0	0.00542	0.286575	20
5		✓	0	0	0.211138	20

The set of eight objects in situation 3 is then extended to 16 objects, by doubling the number of each object, to construct a larger object set. From this new object set, different subsets of 6, 7, 8, 9, and 10 objects, respectively, are randomly generated. A summary of the packing results of 5 random subsets with five random starting positions is in Table 3-2. More details about the results can be found in tables B-1 to B-5 in appendix B.

Table 3-2: Summary of packing results of 5 random subsets with five random starting positions

Subset	Number of objects	Starting position 1	Starting position 2	Starting position 3	Starting position 4	Starting position 5	Feasible solution rate
1	6	✓0.165968	✓0.176289	✓0.16375	✓0.159645	✓0.18837	100%
2	7	✓0.218581	✓0.189769	✗	✓0.190859	✗	60%
3	8	✓0.205351	✗	✗	✓0.194667	✗	40%
4	9	✗	✗	✓0.17428	✓0.163724	✓0.187925	60%
5	10	✗	✗	✗	✗	✗	0%

✓: Output of feasible packing configuration followed by the final value of the fitness function

✗: Output of infeasible packing configuration

When reviewing table 3-2, it is important to observe that from subsets 1-5 the rate of finding a feasible solution decreases except for subset 4. This phenomenon stems from the average size difference of the objects chosen in each subgroup. The objects selected in subset 4 are smaller on average than that of other subsets. The packing results for subsets 1-4 indicate that feasible solutions can be found through multiple attempts, and the best packing solution for each individual subset (each row in table 3-2) can be determined by the lowest fitness value. Whereas for subset 5, it is unlikely to find a feasible solution since multiple attempts with different starting positions did not yield to a feasible solution.

By comparing all the different subsets, it can be observed that, when every object cannot be packed inside the container without constraint violations (packing the 16-object set), the final packing configurations vary greatly, and are closely related to the selection of subsets. In order to find an optimum solution, it is essential to cover the packing results for every possible subgroup of the object set. In the following section, a potential high-level selection algorithm is discussed with the purpose of searching over possible subsets of objects and making wise decisions on which subset to choose.

3.6 Discussion

The proposed packing optimization algorithm tries to pack all the objects inside the container. It will not work when the container is not large enough to pack all of the objects. In this section, a high-level selection algorithm is discussed as a potential solution for selecting subsets of objects and using the packing optimization algorithm to generate the corresponding packing plan. The workflow of the selection algorithm is shown in Figure 3-10.

For a given container, the selection algorithm first determines whether a subset of the total objects set can potentially fit into the container with a reasonable probability based on total volume. Empirically derived density results for types and size distributions of objects can possibly guide this step. Conventional metrics from the concrete aggregates production process (including shape metrics such as elongation, roughness, and flatness) are examples of these types of measures, and they can be acquired automatically in their own right [120–124]. Then the packing optimization algorithm is run multiple times to pack all the objects. If none of the trials are able to find a feasible solution, the selection algorithm chooses another subset and runs the packing optimization algorithm again. Continued failure to pack would result in subsets being selected with smaller numbers of objects. As the number of iterations increase, the selected subset becomes smaller in terms of the number of

objects. In cases of small numbers of objects, the selection algorithm can exhaustively search over all possible subsets. For example, assume n objects are to be packed inside a container. If the packing optimization algorithm cannot find a feasible solution for n objects after multiple runs, the selection algorithm decides to choose a subset of $(n-1)$ objects and conduct multiple runs on each subset. There will be $C_{n-1}^n = n$ different subsets of $n-1$ objects. The selection algorithm continues to select a smaller subset as the program keeps running. In total, the number of subsets will be $C_{n-1}^n + C_{n-2}^n + C_{n-3}^n + \dots + C_1^n = 2^n - 2$. For each subset chosen by the selection algorithm, the packing optimization algorithm is called to generate corresponding packing configurations. The final packing layout will be selected out of all feasible solutions according to the evaluation metric. If the optimizing criterion is the number of packed objects, the feasible solution with the highest number of objects will be chosen. On the other hand, if the packing density is critical, the feasible solution with the highest packing density will be selected as the final output by the selection algorithm.

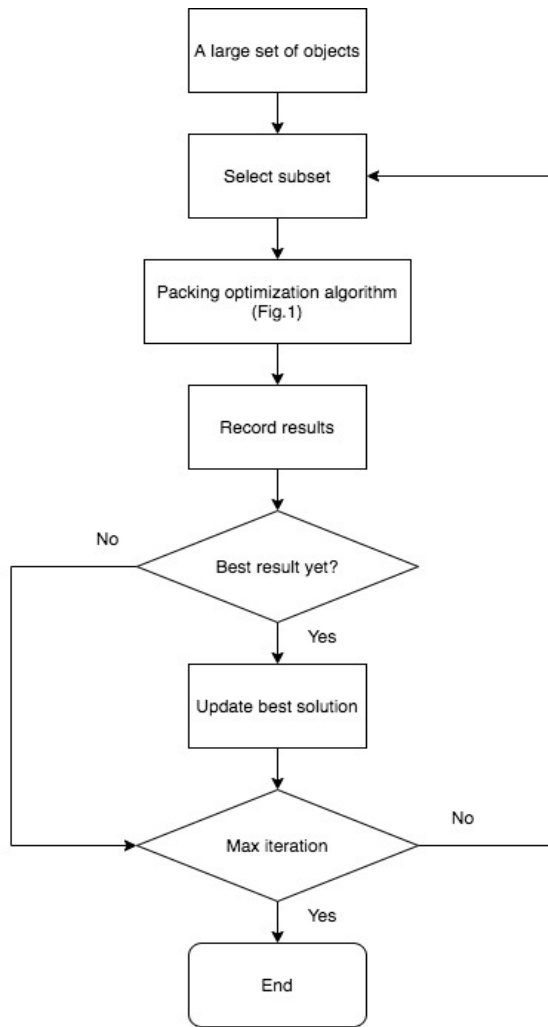


Figure 3-10. Workflow of the selection algorithm

It is not feasible to determine which subsets of objects might fit in the container without attempting to generate each subset's packing configuration. Thus, the selection algorithm runs in iterations with the packing optimization algorithm. In each iteration, the selection algorithm chooses a subset, and the packing optimization algorithm packs the selected objects. The value of the fitness function is compared with the current best solution, and the latter is updated. The selection algorithm stops when a certain number of iterations is reached.

It is worth noting that the exhaustive search selection algorithm is too computationally expensive to try all the possible subset combinations since the number of possible subsets increases exponentially with the number of objects (i.e., NP-hard). The exhaustive search selection algorithm is only

favorable for a problem involving a small number of objects. For a problem with a larger number of objects, a high level heuristic-based algorithm like the one in [84] for allocating objects, or integrating human intervention, could provide better solutions.

3.7 Conclusion and outlook

The practical packing problem of arranging irregular objects (with zero rigid-body deformation) into a box-shaped container with fixed dimensions, defined as "irregular 3D rigid object packing problem", has broad and promising applications. A reasonable methodology incorporating automated data acquisition and packing optimization is presented. Data acquisition involves capturing and pre-processing initial as-is data and generating a mesh representation for each object. Then, simulated annealing is used for packing optimization with continuous translational and rotational degrees of freedom being considered. The methodology is illustrated on various archetypical situations of different scales, ranging from a small set of simple-shaped objects to complex as-is objects. However, the tightly constrained situation reveals the packing optimization algorithm's inefficiency in facing an optimization problem for which the feasible solution is hard to find. The inefficiency of the packing optimization algorithm can be mitigated by choosing a better initial placement. Hence, one future research possibility is to find more effective constraint handling techniques and more advanced metaheuristics.

Preliminary sensitivity analyses were performed to verify the robustness and variations of the packing optimization algorithm, and have also revealed that when the object set is too large to be packed into the container, the final packing configurations vary greatly and are closely related to the selection of subsets. A high-level selection algorithm to exhaustively search over possible subsets of objects is discussed. The selection algorithm illustrates theoretically how an exhaustive search is extremely complicated and time-consuming for a problem with a large number of objects. Integrating human intervention in the selection algorithm, or developing a high-level heuristic-based selection algorithm, are interesting options for future research.

CHAPTER 4

CREATING EFFICIENCY AND MITIGATING RISKS IN TACTICAL WORK PLANNING USING VIRTUAL REALITY ENVIRONMENTS³

4.1 Summary

Due to the inherently dynamic nature of the working environment and accident-prone activities involved, construction is fraught with danger. Accidents not only jeopardize the safety of the workers but also cause work delays. Recent advances in virtual reality (VR) have the potential to support an immersive experience for visualizing complicated and hazardous working environments without exposing workers to potential dangers. This thesis focuses on studying and verifying VR's impact on detailed tactical work planning in terms of reducing risk and improving efficiency compared to conventional methods, specifically for packing optimization in nuclear facility decommissioning and demolition (D&D). VR planning environments are developed in this thesis to demonstrate and enable nuclear waste packing optimization and to minimize risk due to radiation exposure associated with commonly employed manual packing approaches. Experimental results are presented comparing VR-enhanced planning and packing with the conventional methods in terms of packing efficiency and exposure time. Statistical analyses and logical arguments reveal that detailed tactical work planning in an accident-free VR environment allows potential risks and hazards to be identified and mitigated in the planning phase, thus contributing to a safer and more efficient working environment for the human operator.

4.2 Introduction

The construction industry is the largest worldwide. Due to the highly uncertain and accident-prone nature, this sector's projects often exhibit time and cost overruns as well as a poor safety record [125]. From financial planning, resource allocation to time phase planning (scheduling), planning plays a crucial role in construction management and project success. Despite many project planning strategies (e.g., cash flow analysis for financial planning; critical path method, network diagram for scheduling, and minimum moment algorithm, time-cost trade-off analysis for resource allocation), detailed tactical planning of work equipment and procedures before the start of work, even essential, is often

³ This chapter is adapted from Zhao, Y., Jung, Y., Haas, C., & Narasimhan, S. (2021). Creating efficiency and mitigating risks in tactical work planning using virtual reality environments. *Automation in Construction*, Submitted in September 2021, (Under review).

overlooked. Such decisions are not formally planned. The work equipment planning is usually done in the field by toolbox meetings between the foreman and the site manager. The detailed planning of operational procedures is also rarely performed. There are high-risk situations, such as processing power plant outages and optimum packing of nuclear waste, in which there is an unquestionable need to undertake detailed tactical planning beforehand. We would not want the trial-and-error approaches to happen in a radioactive environment, hence a safe and efficient strategy for detailed tactical planning is necessary. By enabling hazard-free virtual interactions between the user and the environment, VR is a powerful tool for detailed tactical planning.

In this thesis, the role and impact of VR implementation on detailed tactical work planning are studied and quantified in terms of risk and efficiency compared to conventional practices. Nearly 70% of the world's nuclear reactors are over 30 years old, and between 200 and 400 reactors are likely to be decommissioned by 2040 [11]. Proactively managing the disposal of nuclear waste could potentially save millions of dollars in decommissioning costs. Based on this need, packing optimization in nuclear facility D&D is presented as an application. Interactive VR planning environments are proposed in this thesis, which facilitates the pre-planning of the packing optimization of nuclear waste through an immersive virtual environment and real-time monitoring of the packing configurations' essential properties. To understand VR-enhanced planning's impact on risk mitigation and efficiency improvement, experiments were conducted to compare the VR planning approaches (human-guided packing and automated packing) with conventional procedures (sequential packing and simulated backhoe loader packing) in terms of packing efficiency and exposure time. VR simulation enables easy testing and evaluation of different robots, interactions, and scenarios without the time-consuming process of setting up complex working situations physically and purchasing expensive machines [126]. The four packing approaches were simulated, tested, and evaluated in virtual environments to fully exploit the VR environments and avoid expensive, time-consuming physical setup of different scenarios. Statistical analyses and logical arguments are presented to justify the potential and effectiveness of applying VR in detailed tactical planning. By investigating and analyzing the application of nuclear waste packing optimization, this thesis also develops and demonstrates an actual implementation of VR-enabled planning with a framework to evaluate and compare different workflows in terms of risk and efficiency for decommissioning applications. Furthermore, the demonstrated VR implementation can be easily adapted for other high-risk applications. To the authors' knowledge, this is the first time VR-enabled planning and its

implementation in packing optimization for nuclear waste have been presented in the literature and are being claimed as the central and original contributions in this thesis.

4.3 Literature Review

Construction planning is a vast topic. Generally speaking, it is the process of budgeting, scheduling, resource allocation, safety planning, and outlining other detailed specifications of the steps to be followed and the constraints to be obeyed in project execution [127]. Good construction planning weights and ensures the technical feasibility of chosen strategies and pursues the efficient utilization of labor, material, and equipment to finish the project on time, on budget, and safely [128]. The planning of a construction project often follows the strategy of decomposition and hierarchical planning, where a complex project is divided into subproblems, and general activities are divided into lower-level activities [129]. Regardless of the different planning levels in hierarchical planning, detailed tactical planning is seldom undertaken for specific work or tasks, leading to the use of trial-and-error approaches in the field. Such a practice is time-consuming, accident-prone, and highly hazardous for high-risk or high investment work, causing great danger for the workers and compromising the project schedule. At the fundamental level, good detailed tactical planning, especially for high-risk and high investment tasks, is key to improving workers' safety and the success rate of projects.

Packing optimization of nuclear waste in nuclear facility D&D is the main application studied in this thesis. Packing optimization problems consist of arranging objects into one or a set of containers to optimize one or multiple objectives such as maximizing the packing efficiency or minimizing the container's volume. The primary constraints of packing optimization problems are that the objects must not overlap with each other and are entirely contained inside the containers [1]. The specific packing problem of relevance to this work involves packing optimization of 3D irregular-shaped objects, referred to as the 3D irregular cutting and packing (C&P) problem. There is a growing interest in 3D irregular C&P problems because of their broad applications and potential impacts in a multitude of industries [10]. Three-dimensional irregular C&P problems can generally be applied both to traditional applications such as improving transport efficiency of building parts or pre-fabricated construction assemblies and emerging applications in civil engineering such as 3D printing in construction and facility waste management [10].

Solution approaches to 3D irregular C&P problems can be classified as belonging to heuristics, metaheuristics, and mathematical programming. Heuristics are rules of thumb that are used to guide, discover and reveal plausible but not necessarily the correct solutions to the problem [72]. For instance, the most popular heuristic algorithm for 3D irregular C&P problems is the Bottom-left-front algorithm, which packs pre-ordered objects one by one at the bottom left front corner of the container's available space [18,71]. Although heuristics are generally relatively fast, they can only explore limited packing configurations. Metaheuristics are combinatorial optimization techniques that provide guidelines to develop a process capable of escaping from local optima and finding a good solution [130]. Genetic algorithm (GA), simulated annealing (SA), Tabu-search are some of the metaheuristics applied to the 3D irregular C&P problem [4,8,46]. Metaheuristics explore more potential configurations resulting in significantly more computational overhead. Researchers have also tried to formulate the 3D irregular C&P problem using mathematical programming. The most successful approach is based on the phi-function, which provides a tool to mathematically describe non-overlapping and containment constraints [7,97]. Heuristics are then applied to reduce the problem into a sequence of subproblems with smaller dimensions and fewer constraints that can be solved using a nonlinear programming solver [7]. The drawback of the phi-function-based mathematical programming method is that it is computationally costly and currently futile for arbitrary shapes. The 3D irregular C&P problems are NP-hard [55]. In other words, the expected time to find an optimal solution increases exponentially as a function of the number of inputs [89]. None of the existing algorithms for the 3D irregular C&P problem can find a globally optimal solution in polynomial time [18]. Overall, finding a suitable solution through autonomous approaches is currently computationally expensive and time-consuming and hence merits investigating a hybrid approach.

Humans can think strategically and are far superior to even state-of-the-art computer algorithms in semantic scene understanding when dealing with visual and spatial data. This means that allowing human intervention in the packing process can decrease the time and computation power required while potentially achieving better outcomes than relying on autonomous packing algorithms alone. VR supports the creation of immersive virtual environments, which simulate a physical environment and allows users to interact with virtually rendered objects in real-time [126,131]. The architecture, engineering, and construction (AEC) industry have witnessed a growing interest in extended reality as a possible solution to certain construction problems in recent years [131–133]. VR has been used in a variety of applications, including training and education, hazard identification, design visualization,

and communication. For instance, Bükürü et al. used VR to create a construction safety training environment that generates precise data to enable individual feedback for the trainees [134]. Kassem et al. presented a systematic literature review regarding VR implementation for safety learning in the construction sector. Their review suggests that risk identification is the most investigated issue in the literature. The research on the subsequent risk management phases, i.e., risk evaluation, risk response planning, and risk monitoring and controlling, is very limited [135]. Muhammad et al. compared the traditional 2D job site layout plan and 3D model in VR for site layout optimization of construction projects [136]. Results indicated that 3D VR-based job site layout planning is more amenable to comprehension by users and enhances collision detection. You et al. employed a VR environment to analyze the safety perception in the human-robot collaborative workspace [126]. Bhandari et al. observed participants interacting with construction hazards in an immersive augmented virtual environment to gain a better understanding of how individuals' emotional states can affect their capacity to perceive hazards, assess risk, and make decisions in the workplace [137]. Alizadehsalehi et al. used a 3D model in an extended reality environment to examine the overall design and evaluate alternative design decisions [133]. To the best of the authors' knowledge, no previous studies have applied VR to detailed tactical task planning, which presents a clear gap in the literature.

VR is proposed for detailed tactical planning in this thesis since it is an intuitive, low-cost, quasi-realistic approach to simulate the working environment and conditions. Through simulations in a VR environment, construction tasks can be easily tested and evaluated to determine the best practices. For the packing optimization application of nuclear waste, VR provides an inexpensive way to harness human intuition and allow workers to perform crucial trial-and-error packing without physically setting up the working environment and being exposed to the real hazard. Moreover, the VR planning environments proposed in this thesis are equipped with a user interface that can instantaneously display essential properties of the ongoing packing configuration, providing critical feedback to workers and guiding better decisions for the ensuing step.

4.4 Methodology

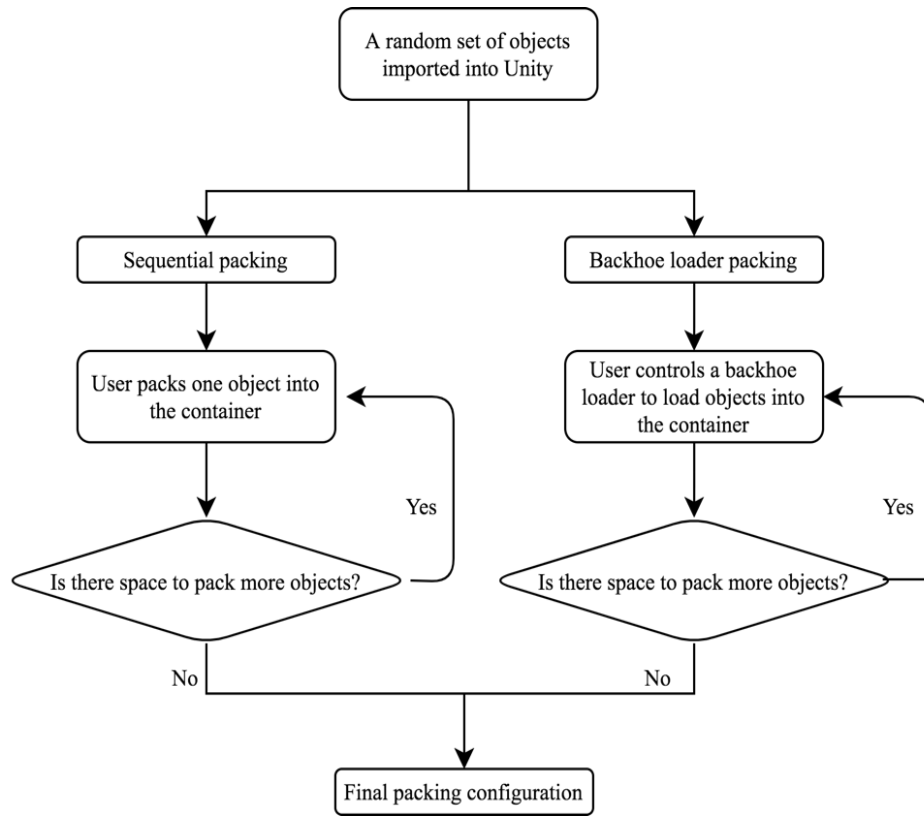
The main premise of this thesis is that VR provides a powerful means to improve planning in construction by mitigating risks and creating efficiency. Specifically, the main objective of the experiments is to demonstrate that: (i) when compared to the conventional approaches, planning using VR can lower potential risks imposed on the worker; (ii) when compared to the conventional approaches, optimization and planning using VR reduces operational errors and increases efficiency.

The description of the experiments and metrics to evaluate the results obtained from the experiments are described next.

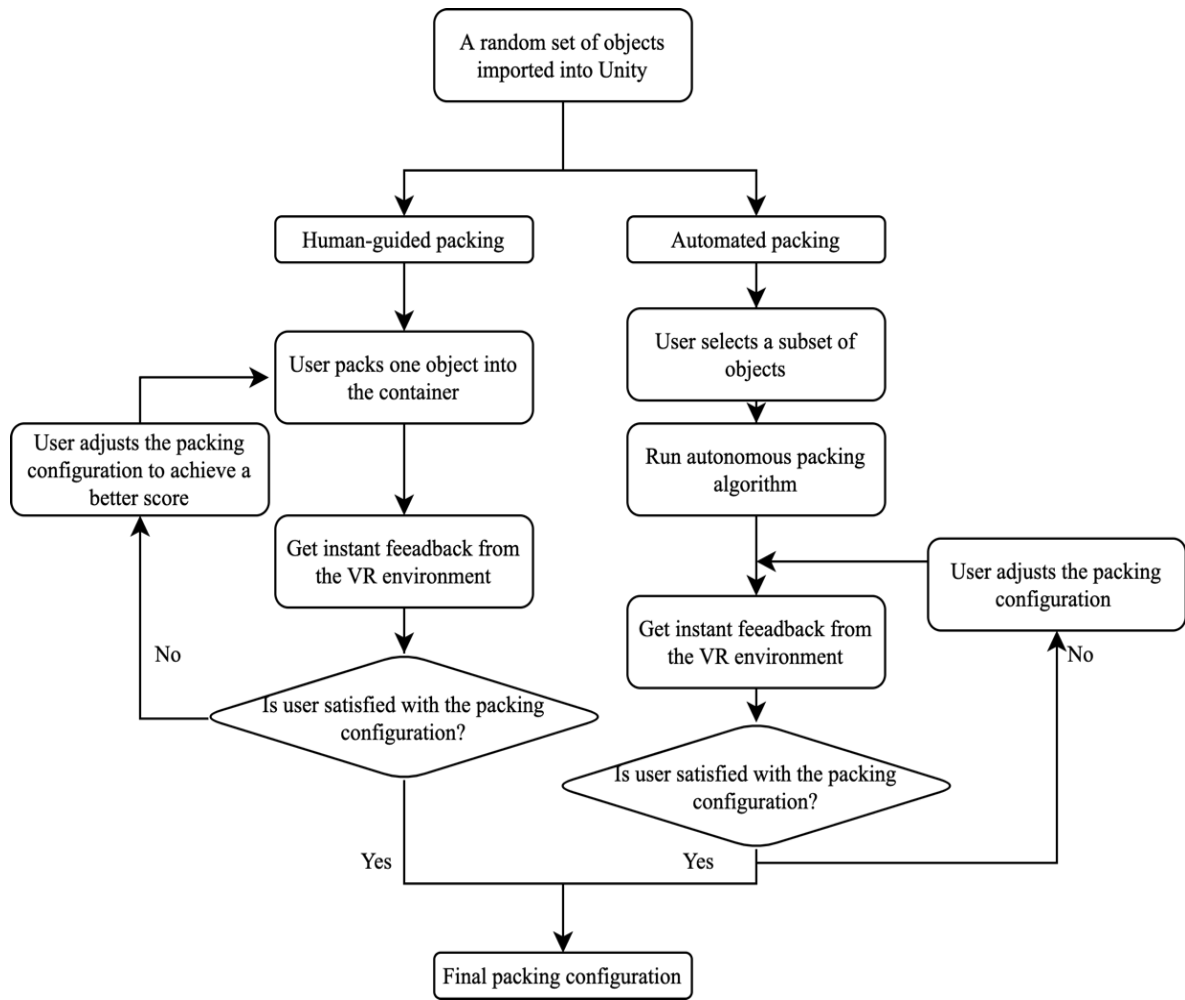
Four packing scenarios were designed in the VR environment which correspond to four distinct packing approaches: (i) sequential packing; (ii) backhoe loader packing; (iii) human-guided packing; and (iv) automated packing. Scenarios (i) and (ii) are considered conventional approaches, while (iii) and (iv) are the newly proposed VR-enhanced planning approaches. VR simulation enables easy testing and evaluation of different robots, interactions, and scenarios without the time-consuming process of setting up complex working situations physically and purchasing expensive machines [126]. The two conventional scenarios were also simulated, and evaluated in VR environments to avoid expensive, time-consuming physical setup of different scenarios. Packing trials were conducted in the four scenarios and results are evaluated and compared from a statistical standpoint. The description of the four packing scenarios are as follows:

- (i) **Sequential packing:** In the VR environment, the user directly handles and places the objects. The user only places objects one at a time until no more objects can be packed inside the container. Once placed inside the container, the object cannot be moved again. The intent is to emulate the stressful, radiation-exposed conditions under which a worker would be performing the work, for which minimizing exposure is paramount.
- (ii) **Backhoe loader packing:** In the VR environment, a user-controlled backhoe loader loads objects into a container. The scenario is intended to simulate objects falling freely and piling up inside the box as a result of gravity when loaded by machinery such as a crane or backhoe. A physics engine models the settling of the objects (gravity assisted). The user stops when no more objects can be packed inside the container.
- (iii) **Human-guided packing:** In the VR environment, the user directly handles and places objects, and receives immediate feedback from the VR user interface. Objects can be moved again after being placed inside the container. The user can adjust the packing configuration as needed until the user is satisfied with the packing configuration. The packing configuration planned in VR planning scenarios can later be executed by the workers or by robots.
- (iv) **Automated packing:** In the VR environment, the user directly handles and places objects, and gets instant feedback from the VR user interface. The user is first required to select a subset of objects to be packed inside the container. An embedded autonomous packing algorithm is available in the VR environment to propose an initial packing configuration to

the user. The user can then manually fine-tune the suggested packing configuration until the user is satisfied with the packing outcome. The packing configuration planned in VR planning scenarios can later be executed by the workers or by robots.



(a)



(b)

Figure 4-1: Workflows for different packing scenarios: (a) conventional approaches. (b) VR planning approaches

The workflows for the four different scenarios are illustrated in Figure 4-1. Sequential packing and backhoe loader packing scenarios use VR only as a simulation tool to mimic real-world approaches without any additional planning steps. Hence, they are referred to as "conventional approaches" in this thesis. They take implicit advantage of the VR environment and its built-in physics engines to avoid physical experiments. Human-guided packing and automated packing scenarios take explicit advantage of the VR environment and adopt VR as an active planning step, which is missing in the conventional approach; these are described as VR planning scenarios in this thesis. Considering the efficiency, operational error, and risk, the following metrics are used to evaluate the outcome from each scenario.

Packing (outcome) efficiency. Packing efficiency is calculated as the ratio of the container volume occupied by the objects and the total container volume, which in other words is the space utilization of the container. Packing outcome efficiency is directly related to the decommissioning cost, especially in nuclear applications where the cost of waste containers can be quite high and not reusable. Packing efficiency measurement depends on a specific situation. Since the literature equates the terms “packing outcome efficiency” with “packing efficiency” the latter term is used throughout the remainder of this thesis.

Weight and radiation limits. Weight and radiation limits are constraints imposed on packing configurations to meet the transportation requirements and the waste acceptance requirements of storage facilities and repositories in nuclear waste packing and storage applications [16,138]. Violation of the two limits results in rework that wastes efforts and time, resulting in a reduction of efficiency and increased risk due to exposure for the workers. These two limits are quantified as a percentage of the container's permissible limit reached by the packing configuration. For example, if the configuration reaches the container's radiation limit, the radiation limitation criterion is 100%. If the weight and radiation limitations are not exceeded, higher values for weight and radiation criteria indicate better container utilization.

Time. For applications such as packing nuclear waste, reducing the overall human exposure time to harmful radiation is the most important risk mitigation measure. The time a user spends on handling objects in the four virtual environments is recorded for each packing configuration and is used for analysis on exposure time in Section 4.6.1.

When packing the nuclear waste objects into a container, the goal is to achieve the highest packing efficiency in the shortest time, by arranging objects into the container without exceeding weight and radiation limits. Participants were pre-trained to go through the learning curve of using the VR system and were determined to have achieved proficiency in all the four scenarios. For each trial, a participant was asked to pack one set of objects under four different scenarios. Forty random sets of objects were generated, resulting in 40 trials in total. Each set, consisting of 15 objects, was randomly selected from a pre-established library of 30 different objects. These objects include broken Styrofoam, concrete debris, and scrap metal pieces. Each object is associated with pre-defined weight and radiation properties to mimic nuclear waste. The weight and radiation properties are assumed to be proportional to the object's volume as a first approximation. All objects were assumed to be made from the same material, whereas objects are randomly assigned to two different radiation intensity

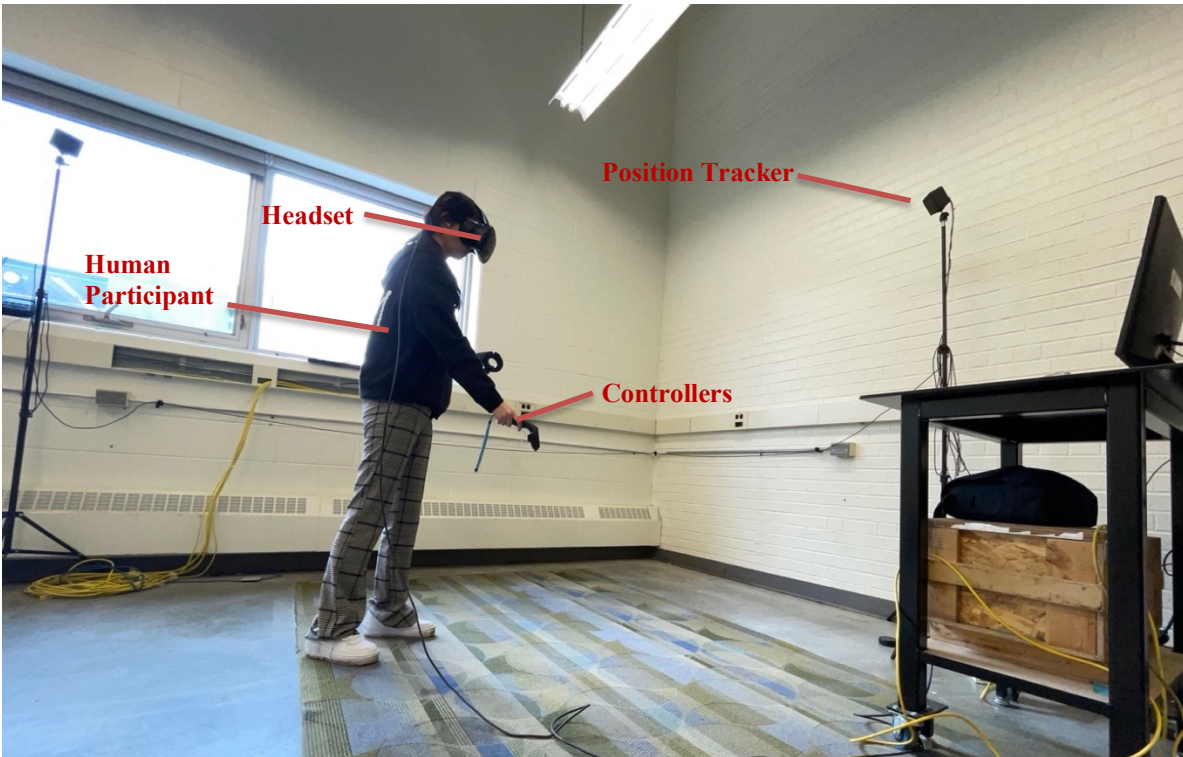
levels. The properties (packing efficiency, radiation limit, weight limit) of final packing configurations from the different scenarios and the time spent performing the activity in the VR environment are all recorded. By comparing the exposure time, the number of invalid packing layouts, and the packing efficiency of valid packing layouts between the four scenarios, VR planning's potential advantages in terms of increasing efficiency and reducing risk were evaluated between the conventional and the VR planning approaches.

4.5 Virtual Packing Environments

The VR packing environments described in this thesis are created using the Unity® game engine [139], which has a built-in physics engine mimicking real-world physics (e.g., gravity and collision). HTC VIVE® [140] is used as the VR headset, as shown in Figure 4-2a. The hardware setup of the VR environment is illustrated in Figure 4-2b. The details regarding the user interface design of the VR packing environments as well as related algorithms can be found in appendix C. The matching controllers are developed with functions such as *grab*, *hold*, and *release* which enable users to manipulate objects such as moving and placing objects into the container in the virtual environment. A pre-established library of 30 objects was created by capturing the initial as-is geometry data for the irregular-shaped objects using the Occipital Structure Sensor [110]. The raw scanned data for the objects require preprocessing using MeshLab® [111] to remove outliers and noise before they can be segmented and converted into a file format (obj) which is compatible with the gaming environment.



(a)



(b)

Figure 4-2. Hardware setting of the VR environments. (a) HTC VIVE® VR kit. (b) Hardware setup for the VR environments

Four virtual packing environments, which correspond to the four different scenarios described previously, feature a container located at the center with an import function to import 15 randomly selected objects from the pre-established library into the packing environment. In the sequential packing scenario, once the objects are imported, the user can start picking up objects and loading them into the container using the controllers. Users can only place each object once and when placed inside the container, the object cannot be moved again. In the backhoe loader packing scenario, a backhoe loader is created using the Mini Tractor asset from the Unity store [141] and the movements of the loader and the front-end bucket are controllable by the user and provide maneuverability and loading ability (into the container), mimicking a real-life backhoe loader.

The two VR planning scenarios (human-guided packing and automated packing) have a similar virtual environment as the sequential packing, with the major difference being the addition of a user interface canvas in the two VR planning scenarios. In the VR planning scenarios, the VR user interface shows values of criteria as real-time feedback to inform the user of the properties of the

current packing configuration so that the user can take packing decisions accordingly. Warning messages are displayed to the user if weight or radiation limitations are exceeded, informing the user of unacceptable packing configurations. In human-guided packing, objects can be moved after being placed inside the container and the users can experiment with different packing configurations to improve packing results.

The automated packing has an integrated packing optimization algorithm to assist the user with packing configuration planning. Details of the autonomous algorithm are introduced in Section 4.5.1. Once the objects are imported into the automated packing environment, the user can select a subset of objects to be packed inside the container. During the selection of objects, the user can rotate the object and identify the objects' preferred orientations. The chosen objects will be highlighted with the preferred orientations, which will be used as their initial orientation in the autonomous algorithm. The user can then start the autonomous algorithm to generate an initial packing configuration, which can then be fine-tuned by the user (manually) if necessary, e.g., to increase efficiency and satisfy constraints if exceeded. Fine-tuning activities available to the user in the automated packing scenario include adjusting the objects' orientations and locations in the packing configuration, remove objects, or add new objects to the packing configuration. Figure 4-3 shows the four virtual packing environments that correspond to the four different scenarios.



(a)

(b)

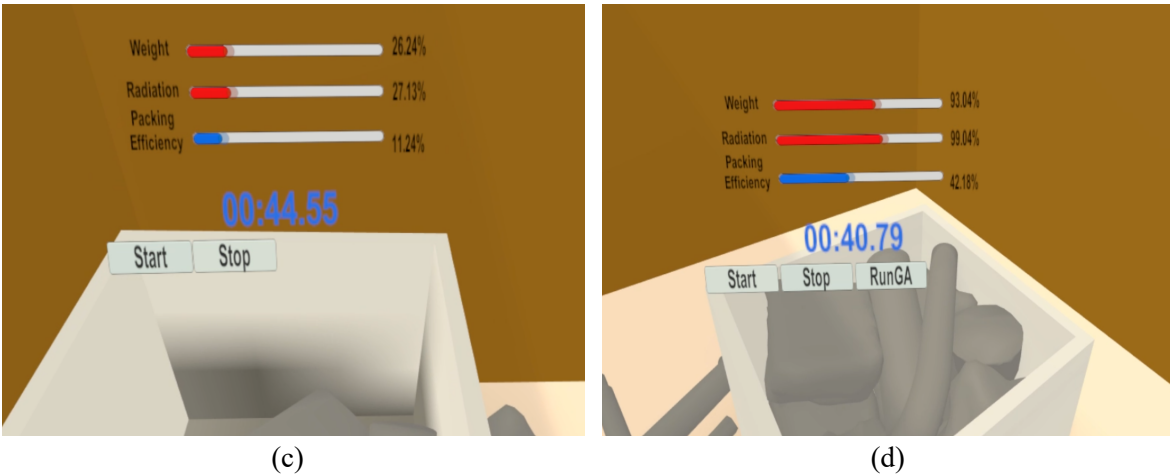


Figure 4-3: Screenshots of the packing environment for (a) sequential packing scenario, (b) backhoe loader packing scenario, (c) human-guided packing scenario, and (d) automated packing scenario

4.5.1. Integrated Heuristic Autonomous Packing Algorithm

An autonomous algorithm is integrated into the automated packing environment for one scenario to provide a good initialization to quickly fine-tune and potentially achieve high packing efficiency. As described in the literature review, metaheuristics algorithms such as GA are popularly used for solving 3D irregular C&P problems. GA is a population-based metaheuristic inspired by the process of natural selection; it broadly follows the steps of generating chromosomes, creating population, selection, crossover reproduction, and mutation. Here, the integrated autonomous packing algorithm is based on the GA-based approach proposed by Gogate [60] and is implemented in C# using the GeneticSharp library [142] with built-in classes for standard GA functions. In the autonomous packing algorithm, GA is used to search for solutions comprising the sequence of packing the objects each with the rotation that leads to a higher packing efficiency. Bottom-left-front (BLF) is implemented to convert each solution to the packing configuration by placing objects one by one at the most bottom left front available space in the container. Since BLF ensures that objects don't overlap during the search process, there will be no collisions in the final packing configuration proposed by the autonomous packing algorithm, leading to easier fine-tuning, if necessary. The pseudo-codes of the BLF and the GA-based autonomous packing algorithm implemented in this chapter are given in Figure 4-4 and 4-5.

In GA, a chromosome is a possible solution to the optimization problem. In the present context, each chromosome conveys information including the packing sequence of the objects and the rotation of each object. The chromosomes are designed as two-dimensional arrays with positive integers. The first row of the 2D array represents the packing sequence of the objects. The next three rows contain binary numbers (0 and 1) representing the rotations of the objects along the Y-axis. For each object, the combination of three binary numbers can represent eight possible rotations from 0 to 315 degrees with an offset of 45 degrees. Bottom-Front-Left (BLF) packing algorithm is implemented as a packing heuristic to finalize objects' translation and convert each chromosome into a packing configuration. For each chromosome in a population, BLF rotates and packs the objects one by one into the bottom left front corner of the container's available space following the sequence and rotation information contained in the chromosome.

The fitness function used to evaluate each chromosome is defined as follows:

$$\text{maximize } f = \begin{cases} e & r \leq 1 \cap w \leq 1 \\ e - 10\% & r > 1 \cup w > 1 \end{cases} \quad (4-1)$$

where, e is the packing efficiency of the current packing configuration, r and w are the radiation limit percentage and weight limit percentage, respectively, achieved by the packing configuration. When the radiation or weight limits are exceeded (i.e., $r > 1$ or $w > 1$), a penalty term is added to the fitness function. In the current nuclear waste packing optimization problem under study, the packing configurations' final packing efficiency ranges from 20% to 45%, according to pre-tests. Subtracting 10% from the packing efficiency results in significantly lower fitness values for corresponding chromosomes when limits are exceeded, ensuring that the algorithm retains and reproduces only good chromosomes. The overall process of integrating GA and BLF in an iterative scheme is given in Figure4-5.

Each population has 50 chromosomes, where the size of the population is derived from the recommendation in [60]. In the reproduction process, elite selection and elite reinsertion were used to retain the best chromosome in the new population during the reproduction, which provides faster convergence. Elite selection orders the evaluated chromosomes based on their fitness values calculated from equation (4-1). Order-based crossover is performed on those ordered parents at a probability of 0.75. When there are fewer offsprings than the minimum population size, the best parents are selected and reinserted in the new generation. To increase the genetic diversity, the flip-bit

mutation was applied. The binary values representing the rotation for the randomly selected object are flipped in the autonomous packing algorithm at a probability of 0.2.

The algorithm selects the best chromosome for each generation. If the best chromosome from this generation has a higher fitness value than the current best chromosome (the best chromosome of all generations), in that case, the best chromosome is updated. The autonomous packing algorithm stops when the max number of generations (pre-defined 30) is reached. A packing configuration with the highest fitness value is proposed to the user in the virtual environment.

```

Begin
For i= 1 to chromosome.Length
{
    Determine currentObject's orientation by rotating object around the y-axis;
    iteration = 1;
    while (iteration < maxIteration)
    {
        if (iteration = 1)
        {
            Place currentObj in the up-right-back available corner of the container;
            if (currentObj is in collision with previously placed objects)
                {break;}
        }
        while (currentObj is not in collision with previously placed objects)
        {
            Move currentObject's position down with an offset yOffset;
        }
        while (currentObj is not in collision with previously placed objects)
        {
            Move currentObject's position left with an offset zOffset;
        }
        while (currentObj is not in collision with previously placed objects)
        {
            Move currentObject's position backwards with an offset xOffset;
        }
        if (no improvement in currentObject's position compared to previous iteration)
            {break;}
        iteration++
    }
}
End

```

Figure 4-4. The pseudo-code for BLF

```

Begin
for  $i = 1$  to  $maxNumberofGenerations$ 
{
  foreach chromosome in population
  { Run BLF to pack the objects following the sequence and rotation information contained in
    chromosome;
    Calculate fitness value of chromosome with Equation (4-1);
  }
  Sort population in ascending order of fitness score;
  for  $j = 1$  to  $populationSize/2$ 
  { Choose the parent chromosomes with best and worst fitness score;
    if ( $crossover\ Rate < randomNumber\ between\ 0\ and\ 1$ )
      {Crossover parent chromosomes at a random point and generate offspring;}
    if ( $mutationRate < randomNumber\ between\ 0\ and\ 1$ )
      {Mutate offspring by randomizing a bit in the array;}
  }
  if  $nextPopulation.Count < populationSize$ 
    {Include best scoring parents until  $nextPopulation.Count == populationSize$ }
  Sort nextPopulation in ascending order of fitness score;
  if ( $nextPopulation[0]$  is better than current bestChromosome)
    {Update bestChromosome}
  Update population;
}
return bestChromosome;
End

```

Figure 4-5: The GA-based autonomous packing algorithm

4.6 Results and Analysis

To demonstrate the two main objectives mentioned in Section 4.4 (lower risks to workers and increase packing efficiency), statistical analysis and logical arguments are presented in this section. The first objective is related to lowering potential risks to workers due to exposure to hazards, which can be evaluated by comparing the workers' implied exposure time in conventional packing scenarios and VR planning scenarios. The second objective is related to avoiding operational errors and

creating efficiency, which can be evaluated by analyzing the number of invalid packing layouts due to violation of the weight or radiation limits and the packing efficiency of valid packing layouts obtained from different scenarios.

4.6.1. Exposure Time

For applications such as packing nuclear waste, reducing the overall human exposure to harmful radiation is the most important risk mitigation measure. To measure this, a timer is implemented in the virtual environments to measure exposure time, which is defined as the total time period users spent in handling and packing nuclear waste into a container. Since users simultaneously plan and execute the packing configuration in conventional approaches, the time spent in each trial in sequential packing and backhoe loader packing scenarios is considered the exposure time. In contrast, users' operation time in the two VR planning scenarios (human-guided packing and autonomous packing) is part of VR planning, thus not counted towards exposure time. The packing configuration planned in VR planning scenarios can later be executed by the workers or by robots. For this study, the former is assumed. In this case, the exposure time of the two VR planning scenarios is the time workers spent executing packing configurations that have been already planned. Participants were asked to execute 40 known pre-planned packing configurations in the VR environments, and the time spent in the execution was recorded. Since the subsequent execution processes of the two VR planning scenarios are the same, the execution time was counted towards exposure time for both VR planning scenarios.

Figure 4-6 illustrates the average time spent in packing the 40 random sets in each scenario. In terms of average exposure time, the backhoe loader packing scenario results in the longest average exposure time since machines are cumbersome to operate and controlling the backhoe loader is not intuitive to the user. Average exposure time in the VR planning approaches is the lowest, directly resulting from the pre-planning process. It is worth noting that the execution of the pre-planned packing configurations can also be completed by robots, which means that there is the potential to reduce the exposure risk even further, to nearly zero. When comparing the two VR planning scenarios, given that users need to select a subset and fine-tune the packing configuration, the average users' planning time in the VR environment is also slightly longer in the autonomous packing scenario. The average total time spent on the autonomous packing scenario is significantly longer than the other three scenarios due to the additional time needed for the autonomous packing algorithm

with the current computing power (The experiments were conducted on a computer with an Intel i7-6700T CPU and 8GB of RAM). Unsurprisingly, the execution time is the same between the two VR enabled approaches since they are both carried out in a near identical fashion once the packing configuration is known from the appropriate scenario.

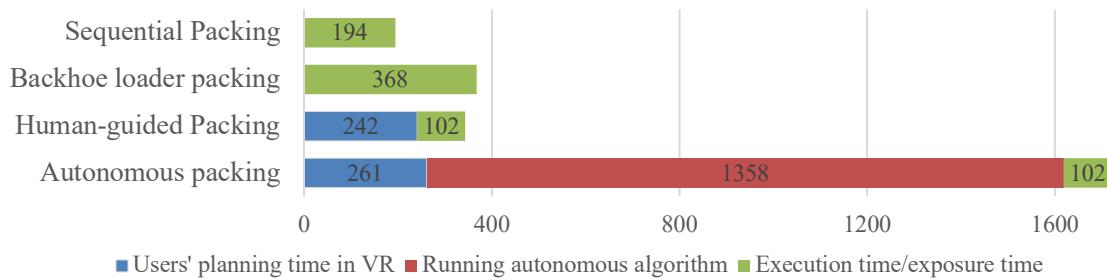


Figure 4-6: Average activity times for different scenarios (unit: s)

4.6.2. Invalid Packing Configurations

Invalid packing configurations refer to configurations where the constraints are violated. More specifically, packing configurations with objects not entirely contained inside the container and packing configurations that exceed the radiation or weight limits are considered invalid. Table 4-1 shows the percentage of the invalid configurations generated in different scenarios. 42.5% of the packing configurations generated in sequential packing scenarios are invalid. All the invalid configurations in the sequential packing scenario were caused by exceeding the weight or radiation limits resulting from the lack of real-time feedback to the user. Meanwhile, the proportion of the invalid packing configurations generated in the backhoe loader packing scenario is 52.5%, which is mainly due to the violation of the containment constraint, i.e., objects are not completely tightly packed inside the container when packed by a backhoe loader based on freefall motion. In the two VR planning scenarios, all the packing configurations are valid, mainly due to the real-time feedback from the user interface in the virtual environment. When the radiation or weight limits are exceeded, warning messages are provided to the user to avoid unqualified packing configurations being generated. Where invalid configurations occur in conventional approaches, workers will need to make adjustments and then repack the objects inside the container until all the constraints are met, resulting in a waste of time and unnecessary radiation exposure to the workers. In this regard, planning in VR significantly reduces human errors, avoids failure and rework, thus creating efficiency, which in turn reduces the risk.

Table 4-1: Number of invalid packing configurations generated in different scenarios

Scenarios	Sequential	Backhoe Loader	Human-guided	Automated
No. of Invalid Packing configurations	17	21	0	0
percentage (%) of Invalid Packing Configurations	42.5%	52.5%	0%	0%

4.6.3. Packing Efficiency

For the valid packing configurations, the packing efficiencies from the four scenarios were evaluated and compared. Statistical analyses were conducted to evaluate the packing efficiency values obtained from four independent packing scenarios (N=114): sequential packing (M=0.3668, SD= 0.02839, n=21), backhoe loader packing (M=0.2724, SD= 0.0418, n=17), human-guided packing (M=0.3946, SD= 0.02414, n=38), and automated packing (M=0.3991, SD= 0.01919, n=38). Two out of the forty object sets happened to be composed of 15 objects of small sizes, resulting in low packing efficiency even for all 15 objects packed inside the container. The packing efficiency results of the two object sets packed under the four scenarios are considered outliers and were not included in this statistical analysis, because they represent situations that would be considered unrealistic. The assumption of normality was evaluated and verified using the Shapiro-Wilk [143] test and is found tenable for all groups at a 0.05 significance level. The assumption of homogeneity of variance was tested using Levene's test; Levene's test for equality of variances was found to be violated, indicating unequal variance among packing efficiency from the four groups. Hence, instead of a one-way analysis of variance, a Welch's F-test [144] was conducted and found to be significant, $F(3, 44.716) = 50.970$, $\text{sig} < 0.001$. Thus, there is significant evidence to conclude that there is a significant difference in the packing efficiency achieved from the four different scenarios.

Post hoc tests were conducted to evaluate the pairwise difference among group means. Given that the equal variance assumption is violated, the Games-Howell test [145] was conducted. The test indicated that the packing efficiency values for human-guided packing and automated packing are not significantly different from each other, with $p = 0.807$ at the significance level of 0.05. However, the test revealed significant pairwise differences between packing efficiencies generated from sequential packing, backhoe loader packing, and the two VR planning scenarios. The mean differences of packing efficiency between sequential packing and the two VR planning scenarios are 2.79% and 3.23%; The mean differences of packing efficiency between backhoe loader packing and the two VR

planning scenarios are 12.22% and 12.67%. The differences show that planning in VR can increase packing efficiency since the VR environment allows humans to explore various packing configurations (assisted by the autonomous packing algorithm or not) without being exposed to hazards. Figure 4-7 illustrates packing efficiencies achieved in different scenarios. The results of the Games-Howell analysis can be found in Table 4-2.

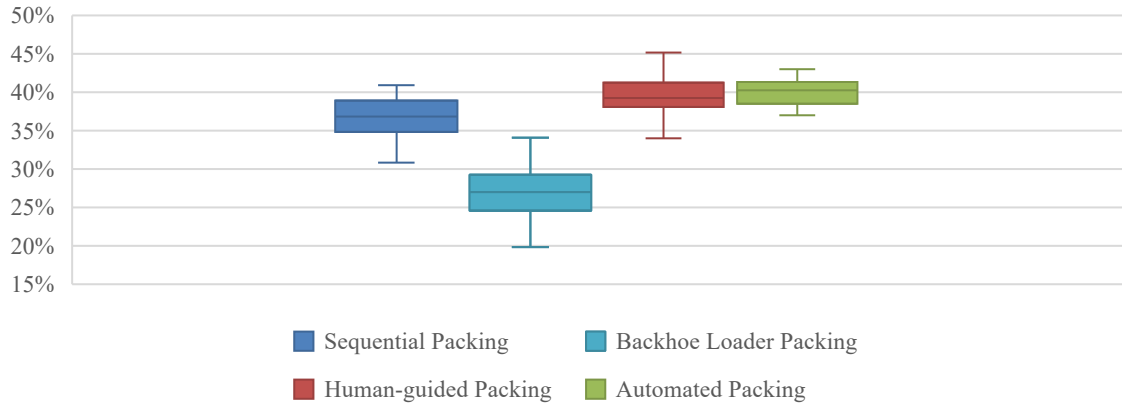


Figure 4-7: Packing efficiencies achieved in different scenarios

Table 4-2: Results from the Games-Howell post hoc test

Scenarios	Scenarios	Mean Difference	Sig.	95% Confidence Interval	
				Lower Bound	Upper Bound
Sequential packing	Backhoe loader packing	0.0943	<0.001*	0.0620	0.1267
	Trial and error	-0.0279	0.002*	-0.0472	-0.0085
	Autonomous packing	-0.0323	<0.001*	-0.0508	-0.0139
Backhoe loader packing	Sequential packing	-0.0943	<0.001*	-0.1267	-0.0620
	Trial and error	-0.1222	<0.001*	-0.1525	-0.0919
	Autonomous packing	-0.1267	<0.001*	-0.1565	-0.0968
Human-guided packing	Sequential packing	0.0279	0.002*	0.0085	0.0472
	Backhoe loader packing	0.1222	<0.001*	0.0919	0.1525
	Autonomous packing	-0.0045	0.807	-0.0176	0.0087
Autonomous packing	Sequential packing	0.0323	<0.001*	0.0139	0.0508
	Backhoe loader packing	0.1267	<0.001*	0.0968	0.1565
	Trial and error	0.0045	0.807	-0.0087	0.0176

*. The mean difference is significant at the 0.05 level.

Compared to conventional approaches, the results show that the VR planning approaches decrease the exposure time, indicating that tactical planning using VR can lower risks imposed on workers in high-risk tasks and hazardous working situations. The lower rate of invalid packing configurations and higher packing efficiency in the VR planning scenarios demonstrate that VR tactical planning can

reduce operational errors, avoid failure, and rework, and create efficiency in high-risk activities such as nuclear waste packing optimization.

The two VR planning scenarios provide comparable packing configurations in terms of packing efficiency. While the autonomous packing scenario requires additional computing time taken up by the autonomous GA-based algorithm (which can be reduced based on the computing resources available), there is relatively little difference in the packing efficiency between the two VR-enabled approaches. The results then seem to suggest that the human-guided scenario is advantageous compared to the autonomous packing scenario, especially since the optimization algorithm adds significant complexity to the process. It should be pointed out this is the case for the relatively small problem size when dealing with a relatively small number of objects. As the problem increases in dimension and complexity, human-guided packing may not scale easily and hence an automated optimization algorithm may become unavoidable. Furthermore, with the availability of high-performance computing, the added cost and computational time required to implement advanced optimization algorithms such as GA has become less of a concern.

Overall, the experiment results demonstrated that the VR environment could be modified as needed to evaluate different scenarios and workflows, providing an ideal tool for detailed task planning before real-world implementation. The investigation of nuclear waste packing optimization application presented the implementation of the VR planning methodology as well as an evaluation system to compare different workflows in terms of risk and efficiency for high-risk tasks.

4.7 Conclusions, Limitations, and Outlook

Even though planning plays a vital role in any project's success in the construction industry, detailed tactical planning of the work equipment and procedures before the start of work, even though especially essential for high-risk tasks and hazardous working environments, is often overlooked. VR is proposed as a safe, flexible, low-cost, powerful tool to conduct detailed tactical planning. To test the impact of the VR enhanced tactical planning, packing optimization of nuclear waste is presented as an application. Experiments were conducted to compare the VR planning approaches and conventional procedures in terms of risk and efficiency. Statistical analyses and logical arguments showed that planning with VR can lower potential risks imposed on workers, avoid failure and rework, create efficiency in the packing processes and improve packing efficiency in the packing configurations. The above-mentioned benefits can result in significant savings in time and money.

Since VR environments can be modified in many ways to adapt different scenarios and workflows, one limitation of this chapter is that the experiment results are partially dependent on the specific workflow implemented, the object set chosen and scanned, and participants. Nonetheless, the experiment results show that the VR environment can be modified as necessary to make useful comparisons between different planned workflows and their potential performances before investing heavily in their implementation in the physical world. It is also worth noting that, although planning with VR improves planning in construction by mitigating risks and creating efficiency, the development of VR planning environments with the assisting features takes time and effort. To use the VR for detailed tactical planning, the time for VR development needs to be planned out beforehand.

The proposed VR interact packing platform utilizes human participation to accelerate the packing process and improve the packing results for the nuclear waste packing optimization problem discussed in this thesis. It considers not only geometry constraints, but also stability and manipulation feasibility constraints. These constraints are checked in the VR environment via its physics engine, which can simulate real-world physics such as gravity and friction. The packing configuration created in the virtual reality environment under real-world physics simulations can then be replicated in the actual world. The VR interactive packing environment also takes capacity constraints (weight and radiation limits) into account, via the user interface canvas and warning messages when constraints are violated. However, there is still much room for improvement in the two VR planning scenarios. More potential improvements remain to be explored, such as a faster autonomous packing algorithm that provides better packing configurations or adding a cutting feature so that large objects can be segmented into smaller objects, which fits better inside a container.

CHAPTER 5

CONCLUSIONS

5.1 Summary and contributions

This thesis aimed to address the emerging need for the 3D irregular cutting and packing (C&P) problems in civil engineering due to their substantial potential impacts in numerous applications, especially optimal packing of nuclear waste generated from the decommissioning and demolition (D&D) of nuclear power plants (NPPs). This research aims to develop algorithms and platforms that generate good packing configurations with multiple objectives in an acceptable time and identify workflows that reduce the risk for human operators in complicated and hazardous working environments.

Considering the two goals, the principal findings of this research effort can be summarized as follows:

Contribution 1. An automated methodology for packing as-is 3D irregular objects.

Chapter 3 presented a methodology incorporating data acquisition and packing optimization for optimal packing of as-is 3D irregular objects. Since nuclear waste from D&D is irregular and does not possess corresponding as-built synthesized models, the methodology utilizes scanned data derived from objects' as-is status to ensure the veracity and reliability of the geometry representations. Then, as an efficient search strategy, a metaheuristic-based packing optimization algorithm, can be used to generate packing configurations. The fitness function, degree of freedom of each object, and the selection metaheuristic can be customized based on specific problems. In Chapter 3, the methodology and its key algorithm are demonstrated to produce effective packing solutions in a mostly automatic manner.

Contribution 2. An interactive and gamified packing platform for 3D irregular C&P problems with multiple objectives

Given the limitations (e.g., computationally demanding and time-consuming) of the fully autonomous packing algorithms, chapter 4 took on a new perspective and presented an interactive packing platform to use the strength of humans (e.g., intuition, strategic thinking, and efficient processing of visual and spatial data) to produce improved outcomes for 3D irregular C&P problems. The interactive packing platform, developed using virtual reality

(VR) and physics engine, enables immersive simulation of real-world physics (e.g., gravity and collision) as well as physical interactions between humans and virtually rendered objects while providing an interactive environment that allowing users to adjust and improve the overall outcome of packing compared to using either the human or fully autonomous packing algorithms alone.

In addition, the proposed VR interactive packing platform takes into account not only geometry constraints, but also stability and manipulation feasibility constraints. These constraints are checked in the VR environment via its physics engine, which can simulate real-world physics such as gravity and friction. The packing configuration created in the virtual reality environment under real-world physics simulations can then be replicated in the actual world. The VR interactive packing environment also takes capacity constraints (weight and radiation limits) into account, via the user interface canvas and warning messages when constraints are violated.

Furthermore, the packing process in the VR packing platform was gamified with a scoring canvas displaying objectives and constraints as real-time feedback to inform the user of the properties of the current packing configuration to make packing decisions accordingly. The gamification of the optimal packing with multiple objectives results in a more engaging and intuitive packing experience and encourages users to improve the packing results.

Contribution 3. Identified and validated VR planning workflows for packing optimization in nuclear facility D&D that release human operators from hazardous working environments and create efficiency

The interactive VR platform presented in Chapter 4 allows users to virtually pack heavy and potentially hazardous objects (e.g., nuclear waste) while limiting exposure to the hazard and physical fatigue; two workflows were identified to minimize risk (due to radiation exposure) and create efficiency during nuclear waste packing optimization using the VR platform. Experiments were conducted to compare the VR enhanced packing planning (human-guided packing and automated packing) with conventional procedures regarding packing efficiency and exposure time. The results revealed that the proposed two workflows with the VR packing platform could reduce operational errors, avoid rework, and create efficiency in nuclear waste packing optimization.

Moreover, the methodology presented in Chapter 4, the tactical work planning using VR environments and the evaluation system for comparison of different workflows in terms of risk and efficiency for high-risk tasks, are not limited to nuclear waste packing optimization; it could potentially extend to numerous construction and manufacturing activities where the potential for hazards to the worker exist.

5.2 Limitations and future research directions

All things considered, the automated methodology with its key packing algorithm, the interactive VR packing platform, and the VR planning workflow form a rational and effective framework for optimal packing of 3D irregular objects in civil engineering applications, especially in applications like nuclear waste packing optimization when hazards are present. However, many methodological and technical challenges remain, which still warrant further attention in future research efforts. Certain limitations of this research are discussed, particularly in the context of directions for future research:

1. Improvement of the existing autonomous packing algorithms

One limitation of the automated methodology for optimal packing of as-is 3D irregular objects, presented in Chapter 4, is its packing optimization algorithm's inefficiency in facing an optimization problem for which the feasible solution is hard to find because it is so tightly constrained. Future research should improve autonomous approaches' applicability by mitigating the inefficiency and speeding up the computation time.

Apart from the heuristic-based algorithms discussed in Chapter 2, high-level strategies can be used to improve packing results and accelerate the packing process. For example, prior to the packing process, objects can be studied to uncover hidden patterns that may help improve the packing outcomes. To optimise the packing outcomes, objects might be grouped according to their geometry similarity or their size gradations. Prior to the packing process, the size distribution of the objects can be examined; an optimum size distribution of the objects may be discovered, resulting in increased packing efficiency. Civil engineering offers similar analogies in which grain size distribution affects the packing density and packing performance. For instance, different grain size distribution of soils can result in different dry densities and have a substantial effect on the soil's macro-scale behaviors as well as macro performance characteristics such as shear strength [146]. Similarly, the grain size distribution of the aggregates can affect both manufacture and field performance of asphalt since asphalt

relies on aggregate contact to withstand loadings in pavement engineering [147,148]. Apart from grouping objects prior to packing, strategies can be applied after packing configurations are generated. Researchers discovered that shaking after packing might gradually improve packing efficiency [149]. By incorporating shaking and vibration movement after the packing process, denser packing configurations can be achieved.

The improvement can be also achieved by exploring and incorporating innovative technologies. For instance, hybridizing different search techniques, such as developing heuristics based on mathematical programming models, i.e., matheuristics, is an important research avenue. Moreover, with recent advancements, reinforcement learning (RL) has been applied to combinatorial optimization problems like traveling salesman and knapsack problems. Some researchers have implemented RL to bin packing problems. The development of RL approaches for 3D irregular C&P problems has great potential as well.

2. Variants of 3D irregular C&P problem in civil engineering

Algorithms, platforms, and workflows for optimal packing of as-is 3D irregular objects were presented in this research. However, many other closely related problems remain unresolved in this dissertation, posing new challenges and opportunities.

In the case of optimal packing problems involving a large number of objects, the objects must be allocated into a set of containers in order to maximize overall packing efficiency or minimize the number of required containers. Future research in multi-container 3D irregular C&P problems will require efficient allocation algorithms that optimally group objects into different containers for better packing results. The selection algorithm presented in Chapter 3 demonstrates theoretically how an exhaustive search is extremely complicated and time-consuming. As a result, one future research possibility is to develop more efficient allocation techniques, such as a heuristic-based allocation algorithm.

Finding an optimized way to decompose structures or large objects into parts that fit inside containers can also improve the packing efficiency of the subsequent packing. The decompose-and-pack problem is the name given to this problem. The decompose-and-pack problem seeks to decompose components or structures into packable parts, which can then be efficiently packed, in addition to optimizing the packing of as-is objects. The combination of decomposition and packing creates both challenges and research opportunities.

3. Improvement on the interactive VR packing platform and VR-enhanced planning approach

Following the aforementioned unexplored problems, new features for the VR packing platform presented in Chapter 4 can be implemented, including segmenting large objects into smaller parts that fit more efficiently inside a container, simulating object deformation if objects need to be crushed before packing [150], and adding shaking movement to improve packing efficiency.

Furthermore, one limitation of the current VR packing platform is that objects in the platform are assigned with pre-defined weight and radiation properties in order to simulate real-world nuclear waste. A new research avenue opens up when building information modelling (BIM) and virtual reality (VR) are combined. Because of BIM's ability to integrate data from multiple sources, comprehensive and data-rich structure models created with BIM can be imported into VR, with required information better presented in an immersive virtual environment. This data can then be used to label and categorize structures for subsequent decompose and packing optimization.

Chapter 4 also demonstrated that detailed tactical work planning using VR environments could reduce risk and improve efficiency compared to traditional methods, particularly for high-risk, high-investment tasks. More potential applications and research avenues for VR-enhanced planning must be explored. For instance, robotics and automation can help VR-enhanced technical planning reduce potential risks for human operators during the data collection and execution phases. A remote decommissioning and demolition methodology that includes a robot platform for rapid inspection, a 3D model for data archiving and demonstration and a VR platform for waste segmentation and packing optimization can be developed. Additionally, a Charrette test [151] might be undertaken in the future to evaluate the influence of VR environments from the user experience perspective prior to implementing detailed tactical work planning in VR environments.

5.3 Publications

The peer-refereed publications, directly related to the scope of this thesis, and authored by the candidate are listed below:

5.3.1. Peer-refereed journal articles

1. **Zhao, Y.**, Rausch, C., & Haas, C. (2021). Optimizing 3D irregular object packing from 3D scans using metaheuristics. *Advanced Engineering Informatics*, 47, 101234. DOI: <https://doi.org/10.1016/j.aei.2020.101234>
2. **Zhao, Y.**, Jung, Y., Haas, C., & Narasimhan, S. (2021). Creating efficiency and mitigating risks in tactical work planning using virtual reality environments. *Automation in Construction*, Submitted in September 2021. (Under review)
3. **Zhao, Y.**, Haas, C., & Narasimhan, S. (2021). A survey of the literature on 3d irregular C&P problems. *Journal of Computing in Civil Engineering*, Submitted in November 2021. (Under review)
4. **Zhao, Y.**, Earle, G., Ferhatoglu, E., Jung, Y., Morales, D., Haas, C., & Narasimhan, S. (2021). Human-robot collaborative workflow for remote decommissioning and demolition. (In preparation).

5.3.2. Peer-refereed conference papers

5. **Zhao, Y.**, Jung, Y., Haas, C., & Narasimhan, S. (2021). Virtual Reality Platform for 3D Irregular Packing Problem. In 28th International Workshop on Intelligent Computing in Engineering (pp. 281-290).
6. Rausch, C., **Zhao, Y.**, & Haas, C. (2020). Parametric or Non-parametric? Understanding the Inherent Trade-offs between Forms of Object Representation. In ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction (Vol. 37, pp. 248-255). IAARC Publications.
7. **Zhao, Y.**, & Haas, C. T. (2019). A 3D Irregular Packing Algorithm Using Point Cloud Data. In *Computing in Civil Engineering 2019: Data, Sensing, and Analytics* (pp. 201-208). Reston, VA: American Society of Civil Engineers.

Bibliography

- [1] A.A.S. Leao, F.M.B. Toledo, J.F. Oliveira, M.A. Carravilla, R. Alvarez-Valdés, Irregular packing problems: A review of mathematical models, *Eur. J. Oper. Res.* (2019). <https://doi.org/10.1016/j.ejor.2019.04.045>.
- [2] Y. Stoyan, A. Pankratov, T. Romanova, Cutting and packing problems for irregular objects with continuous rotations: Mathematical modelling and non-linear optimization, *J. Oper. Res. Soc.* 67 (2016) 786–800. <https://doi.org/10.1057/jors.2015.94>.
- [3] L. Araújo, E. Ozcan, J. Atkin, M. Baumann, C. Tuck, R. Hague, Toward Better Build Volume Packing in Additive Manufacturing: Classification of Existing Problems and Benchmarks, *Proc. Solid Free. Fabr. Symp.* (2015) 401–410.
- [4] J. Vanek, J.A.G. Galicia, B. Benes, R. Měch, N. Carr, O. Stava, G.S. Miller, PackMerger: A 3D print volume optimizer, *Comput. Graph. Forum.* 33 (2014) 322–332. <https://doi.org/10.1111/cgf.12353>.
- [5] J. Egeblad, B.K. Nielsen, A. Odgaard, Fast neighborhood search for two- and three-dimensional nesting problems, *Eur. J. Oper. Res.* 183 (2007) 1249–1266. <https://doi.org/10.1016/j.ejor.2005.11.063>.
- [6] G. Wäscher, H. Haußner, H. Schumann, An improved typology of cutting and packing problems, *Eur. J. Oper. Res.* 183 (2007) 1109–1130. <https://doi.org/10.1016/j.ejor.2005.12.047>.
- [7] T. Romanova, J. Bennell, Y. Stoyan, A. Pankratov, Packing of concave polyhedra with continuous rotations using nonlinear optimisation, *Eur. J. Oper. Res.* 268 (2018) 37–53. <https://doi.org/10.1016/j.ejor.2018.01.025>.
- [8] M. Fakoor, S.M.N. Ghoreishi, H. Sabaghzadeh, Spacecraft Component Adaptive Layout Environment (SCALE): An efficient optimization tool, *Adv. Sp. Res.* 58 (2016) 1654–1670. <https://doi.org/10.1016/j.asr.2016.07.020>.
- [9] S. Szykman, J. Cagan, A simulated annealing-based approach to three-dimensional component packing, *J. Mech. Des. Trans. ASME.* 117 (1995) 308–314. <https://doi.org/10.1115/1.2826140>.

- [10] Y. Zhao, C. Rausch, C. Haas, Optimizing 3D Irregular Object Packing from 3D Scans Using Metaheuristics, *Adv. Eng. Informatics*. 47 (2021) 101234.
<https://doi.org/10.1016/j.aei.2020.101234>.
- [11] D. IURCHAK, 200 – 400 Nuclear reactors to be decommissioned by 2040, (2021) 1–8.
<https://energypost.eu/200-400-nuclear-reactors-to-be-decommissioned-by-2040/> (accessed May 21, 2021).
- [12] IEA, *Nuclear Power in a Clean Energy System*, IEA, Paris, 2019.
<https://doi.org/10.1787/fc5f4b7e-en>.
- [13] Organisation for Economic Co-operation and Development, *Decommissioning Nuclear Power Plants: Policies, Strategies and Costs*, 2003. <https://doi.org/10.4324/9780429042447-11>.
- [14] G. Manon, Besnard, Marcos, Buser, Ian, Fairlie, *The World Nuclear Waste Report 2019*, 2019.
- [15] Q. Deng, S. Zou, H. Chen, W. Duan, Research on visualization and error compensation of demolition robot attachment changing, *Sensors (Switzerland)*. 20 (2020) 1–16.
<https://doi.org/10.3390/s20082428>.
- [16] INTERNATIONAL ATOMIC ENERGY AGENCY, *Disposal Aspects of Low and Intermediate Level Decommissioning Waste*, IAEA, Vienna, 2008.
- [17] Natural Resources Canada, *Inventory of Radioactive Waste in Canada*, 2016.
https://www.nrcan.gc.ca/sites/www.nrcan.gc.ca/files/energy/pdf/uranium-nuclear/17-0467_Canada_Radioactive_Waste_Report_access_e.pdf.
- [18] L.J.P. Araújo, A. Panesar, E. Özcan, J. Atkin, M. Baumers, I. Ashcroft, An experimental analysis of deepest bottom-left-fill packing methods for additive manufacturing, *Int. J. Prod. Res.* 7543 (2019). <https://doi.org/10.1080/00207543.2019.1686187>.
- [19] P. Cao, Z. Fan, R.X. Gao, J. Tang, Harnessing multi-objective simulated annealing toward configuration optimization within compact space for additive manufacturing, *Robot. Comput. Integr. Manuf.* 57 (2019) 29–45. <https://doi.org/10.1016/j.rcim.2018.10.009>.
- [20] S. Szykman, J. Cagan, Constrained three-dimensional component layout using simulated annealing, *J. Mech. Des. Trans. ASME*. 119 (1997) 28–35. <https://doi.org/10.1115/1.2828785>.
- [21] X. Zhang, B. Zhou, Y. Zeng, P. Gu, Model layout optimization for solid ground curing rapid

- prototyping processes, *Robot. Comput. Integr. Manuf.* 18 (2002) 41–51.
[https://doi.org/10.1016/S0736-5845\(01\)00022-9](https://doi.org/10.1016/S0736-5845(01)00022-9).
- [22] S.M. Hur, K.H. Choi, S.H. Lee, P.K. Chang, Determination of fabricating orientation and packing in SLS process, *J. Mater. Process. Technol.* 112 (2001) 236–243.
[https://doi.org/10.1016/S0924-0136\(01\)00581-7](https://doi.org/10.1016/S0924-0136(01)00581-7).
- [23] H.R. Thomas, D.R. Riley, J.I. Messner, *Fundamental Principles of Site Material Management*, *J. Constr. Eng. Manag.* 131 (2005) 808–815. <https://doi.org/10.1061/9780784414651.ch06>.
- [24] B. Anvari, P. Angeloudis, W.Y. Ochieng, A multi-objective GA-based optimisation for holistic Manufacturing, transportation and Assembly of precast construction, *Autom. Constr.* 71 (2016) 226–241. <https://doi.org/10.1016/j.autcon.2016.08.007>.
- [25] A. Al Rashid, S.A. Khan, S. G. Al-Ghamdi, M. Koç, Additive manufacturing: Technology, applications, markets, and opportunities for the built environment, *Autom. Constr.* 118 (2020) 103268. <https://doi.org/10.1016/j.autcon.2020.103268>.
- [26] V. Thangavelu, Y. Liu, M. Saboia, N. Napp, Dry Stacking for Automated Construction with Irregular Objects, *Proc. - IEEE Int. Conf. Robot. Autom.* (2018) 4782–4789.
<https://doi.org/10.1109/ICRA.2018.8460562>.
- [27] Y. Oh, C. Zhou, S. Behdad, Part decomposition and assembly-based (Re) design for additive manufacturing: A review, *Addit. Manuf.* 22 (2018) 230–242.
<https://doi.org/10.1016/j.addma.2018.04.018>.
- [28] L.J.P. Araújo, E. Özcan, J.A.D. Atkin, M. Baumers, Analysis of irregular three-dimensional packing problems in additive manufacturing: a new taxonomy and dataset, *Int. J. Prod. Res.* 57 (2019) 5920–5934. <https://doi.org/10.1080/00207543.2018.1534016>.
- [29] J. Cagan, K. Shimada, S. Yin, A survey of computational approaches to three-dimensional layout problems, *CAD Comput. Aided Des.* 34 (2002) 597–611.
[https://doi.org/10.1016/S0010-4485\(01\)00109-9](https://doi.org/10.1016/S0010-4485(01)00109-9).
- [30] H. Dyckhoff, A typology of cutting and packing problems, *Eur. J. Oper. Res.* 44 (1990) 145–159. [https://doi.org/10.1016/0377-2217\(90\)90350-K](https://doi.org/10.1016/0377-2217(90)90350-K).
- [31] G. Belov, G. Scheithauer, A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths, *Eur. J. Oper. Res.* 141 (2002) 274–294.

[https://doi.org/10.1016/S0377-2217\(02\)00125-X](https://doi.org/10.1016/S0377-2217(02)00125-X).

- [32] F.K. Lemos, A.C. Cherri, S.A. de Araujo, The cutting stock problem with multiple manufacturing modes applied to a construction industry, *Int. J. Prod. Res.* (2020). <https://doi.org/10.1080/00207543.2020.1720923>.
- [33] M. Gradišar, M. Kljajić, G. Resinovič, J. Jesenko, A sequential heuristic procedure for one-dimensional cutting, *Eur. J. Oper. Res.* 114 (1999) 557–568. [https://doi.org/10.1016/S0377-2217\(98\)00140-4](https://doi.org/10.1016/S0377-2217(98)00140-4).
- [34] A. Al Hawarneh, S. Bendak, F. Ghanim, Construction site layout planning problem: Past, present and future, *Expert Syst. Appl.* (2020) 114247. <https://doi.org/10.1016/j.eswa.2020.114247>.
- [35] J.J. Michalek, R. Choudhary, P.Y. Papalambros, Architectural layout design optimization, *Eng. Optim.* 34 (2002) 461–484. <https://doi.org/10.1080/03052150214016>.
- [36] A.M. El Ansary, M.F. Shalaby, Evolutionary optimization technique for site layout planning, *Sustain. Cities Soc.* 11 (2014) 48–55. <https://doi.org/10.1016/j.scs.2013.11.008>.
- [37] G. Fasano, A global optimization point of view to handle non-standard object packing problems, *J. Glob. Optim.* 55 (2013) 279–299. <https://doi.org/10.1007/s10898-012-9865-8>.
- [38] X. Chen, W. Yao, Y. Zhao, X. Chen, W. Liu, A novel satellite layout optimization design method based on phi-function, *Acta Astronaut.* 180 (2020) 560–574. <https://doi.org/10.1016/j.actaastro.2020.12.034>.
- [39] J.N. Zheng, C.F. Chien, M. Gen, Multi-objective multi-population biased random-key genetic algorithm for the 3-D container loading problem, *Comput. Ind. Eng.* 89 (2015) 80–87. <https://doi.org/10.1016/j.cie.2014.07.012>.
- [40] Y. Ma, Z. Chen, W. Hu, W. Wang, Packing Irregular Objects in 3D Space via Hybrid Optimization, *Comput. Graph. Forum.* 37 (2018) 49–59. <https://doi.org/10.1111/cgf.13490>.
- [41] D. Meagher, Geometric modeling using octree encoding, *Comput. Graph. Image Process.* 19 (1982) 129–147. [https://doi.org/10.1016/0146-664X\(82\)90104-6](https://doi.org/10.1016/0146-664X(82)90104-6).
- [42] M. Boccia, S. di Muro, F. Mosca, A. Sforza, C. Sterle, A fast heuristic for a three-dimensional non-convex domain loading problem, *4or.* 9 (2011) 83–101. <https://doi.org/10.1007/s10288-010-0133-9>.

- [43] M. Verkhoturov, A. Petunin, G. Verkhoturova, K. Danilov, D. Kurennov, The 3D Object Packing Problem into a Parallelepiped Container Based on Discrete-Logical Representation, *IFAC-PapersOnLine*. 49 (2016) 1–5. <https://doi.org/10.1016/j.ifacol.2016.07.540>.
- [44] M. Attene, Shapes in a Box: Disassembling 3D Objects for Efficient Packing and Fabrication, *Comput. Graph. Forum*. 34 (2015) 64–76. <https://doi.org/10.1111/cgf.12608>.
- [45] J. Cagan, D. Degentesh, S. Yin, A simulated annealing-based algorithm using hierarchical models for general three-dimensional component layout, *CAD Comput. Aided Des.* 30 (1998) 781–790. [https://doi.org/10.1016/S0010-4485\(98\)00036-0](https://doi.org/10.1016/S0010-4485(98)00036-0).
- [46] E. Stefan, W. Paul, Packing Irregular-Shaped Objects for 3D Printing, (2015) 3–15. <https://doi.org/10.1007/978-3-319-24489-1>.
- [47] V. Chekanin, Solving the problem of packing objects of complex geometric shape into a container of arbitrary dimension, *CEUR Workshop Proc.* 2744 (2020) 1–13. <https://doi.org/10.51130/graphicon-2020-2-3-50>.
- [48] X. Chen, H. Zhang, J. Lin, R. Hu, L. Lu, Q. Huang, B. Benes, D. Cohen-Or, B. Chen, Dapper: Decompose-and-pack for 3D printing, *ACM Trans. Graph.* 34 (2015) 1–12. <https://doi.org/10.1145/2816795.2818087>.
- [49] Y.G. Stoyan, N.I. Gil, G. Scheithauer, A. Pankratov, I. Magdalina, Packing of convex polytopes into a parallelepiped, *Optimization*. 54 (2005) 215–235. <https://doi.org/10.1080/02331930500050681>.
- [50] N. Chernov, Y. Stoyan, T. Romanova, Mathematical model and efficient algorithms for object packing problem, *Comput. Geom.* 43 (2010) 535–553. <https://doi.org/10.1016/j.comgeo.2009.12.003>.
- [51] Y. Stoyan, A. Pankratov, T. Romanova, Quasi-phi-functions and optimal packing of ellipses, *J. Glob. Optim.* 65 (2016) 283–307. <https://doi.org/10.1007/s10898-015-0331-2>.
- [52] J. Egeblad, D. Pisinger, Heuristic approaches for the two- and three-dimensional knapsack packing problem, *Comput. Oper. Res.* 36 (2009) 1026–1049. <https://doi.org/10.1016/j.cor.2007.12.004>.
- [53] S.R. Sukumar, D.L. Page, A.F. Koschan, M.A. Abidi, Towards understanding what makes 3D objects appear simple or complex, 2008 IEEE Comput. Soc. Conf. Comput. Vis. Pattern

- Recognit. Work. CVPR Work. (2008) 1–8. <https://doi.org/10.1109/CVPRW.2008.4562975>.
- [54] E. Pei, R.I. Campbell, D. De Beer, Entry-level RP machines: How well can they cope with geometric complexity?, *Assem. Autom.* 31 (2011) 153–160. <https://doi.org/10.1108/01445151111117737>.
- [55] A. Bortfeldt, G. Wäscher, Constraints in container loading-A state-of-the-art review, *Eur. J. Oper. Res.* 229 (2013) 1–20. <https://doi.org/10.1016/j.ejor.2012.12.006>.
- [56] F. Wang, K. Hauser, Stable bin packing of non-convex 3D objects with a robot manipulator, *Proc. - IEEE Int. Conf. Robot. Autom.* 2019-May (2019) 8698–8704. <https://doi.org/10.1109/ICRA.2019.8794049>.
- [57] Y.S. Wang, H.F. Teng, Y.J. Shi, Cooperative co-evolutionary scatter search for satellite module layout design, *Eng. Comput. (Swansea, Wales)*. 26 (2009) 761–785. <https://doi.org/10.1108/02644400910985161>.
- [58] J. Egeblad, C. Garavelli, S. Lisi, D. Pisinger, Heuristics for container loading of furniture, *Eur. J. Oper. Res.* 200 (2010) 881–892. <https://doi.org/10.1016/j.ejor.2009.01.048>.
- [59] S.H. Jang, K.Y. Rhee, An application of annealing algorithm to the layout design of ocean space submersible boat, *JSME Int. Journal, Ser. C Mech. Syst. Mach. Elem. Manuf.* 47 (2004) 770–775. <https://doi.org/10.1299/jsmec.47.770>.
- [60] A.S. Gogate, S.S. Pande, Intelligent layout planning for rapid prototyping, *Int. J. Prod. Res.* 46 (2008) 5607–5631. <https://doi.org/10.1080/00207540701277002>.
- [61] A. Chehouri, R. Younes, J. Perron, A. Ilinca, A constraint-handling technique for genetic algorithms using a violation factor, *J. Comput. Sci.* 12 (2016) 350–362. <https://doi.org/10.3844/jcssp.2016.350.362>.
- [62] X. Liu, J. Liu, A. Cao, Z. Yao, HAPE3D—a new constructive algorithm for the 3D irregular packing problem, *Front Inf. Technol Electron Eng.* 16 (2015) 380–390. <https://doi.org/10.1631/FITEE.1400421>.
- [63] S. Yin, J. Cagan, An extended pattern search algorithm for three-dimensional component layout, *J. Mech. Des. Trans. ASME.* 122 (2000) 102–108. <https://doi.org/10.1115/1.533550>.
- [64] S. Tiwari, G. Fadel, P. Fenyés, A fast and efficient compact packing algorithm for SAE and ISO luggage packing problems, *J. Comput. Inf. Sci. Eng.* 10 (2010) 1–11.

<https://doi.org/10.1115/1.3330440>.

- [65] J.K. Dickinson, G.K. Knopf, Packing subsets of 3D parts for layered manufacturing, *Int. J. Smart Eng. Syst. Des.* 4 (2002) 147–161. <https://doi.org/10.1080/10255810213478>.
- [66] J.J.K. Dickinson, G.K.G. Knopf, Serial packing of arbitrary 3D objects for optimizing layered manufacturing, *Proc. SPIE.* 3522 (1998) 130–138. <https://doi.org/10.1117/12.325756>.
- [67] Y.-K. Joung, S. Do Noh, Intelligent 3D packing using a grouping algorithm for automotive container engineering, *J. Comput. Des. Eng.* 1 (2014) 140–151. <https://doi.org/10.7315/jcde.2014.014>.
- [68] J. Egeblad, B.K. Nielsen, M. Brazil, Translational packing of arbitrary polytopes, *Comput. Geom. Theory Appl.* 42 (2009) 269–288. <https://doi.org/10.1016/j.comgeo.2008.06.003>.
- [69] J. Hur, K. Lee, J. Ahn, H.C. Lee, A three-dimensional algorithm using two-dimensional slice data for building multiple parts in layered manufacturing, *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.* 214 (2000) 365–378. <https://doi.org/10.1243/0954405001518152>.
- [70] W. Yang, W. Liu, L. Liu, A. Xu, A genetic algorithm for automatic packing in rapid prototyping processes, *Int. Conf. Intell. Comput.* 5226 LNCS (2008) 1072–1077. https://doi.org/10.1007/978-3-540-87442-3_132.
- [71] S. Wu, M. Kay, R. King, A. Vila-Parrish, D. Warsing, Multi-objective optimization of 3D packing problem in additive manufacturing, in: *IIE Annu. Conf. Expo 2014*, 2014: pp. 1485–1494. <https://search.proquest.com/docview/1622299299?pq-origsite=gscholar> (accessed January 3, 2018).
- [72] B. V. Koen, Toward a Definition of the Engineering Method, *Eur. J. Eng. Educ.* 13 (1988) 307–315. <https://doi.org/10.1080/03043798808939429>.
- [73] J.K. Dickinson, G.K. Knopf, A Moment based metric for 2-D and 3-D packing, *Eur. J. Oper. Res.* 122 (2000) 133–144. [https://doi.org/10.1016/S0377-2217\(99\)00061-2](https://doi.org/10.1016/S0377-2217(99)00061-2).
- [74] H. fei Teng, S. lin Sun, D. quan Liu, Y. zhao Li, Layout optimization for the objects located within a rotating vessel - A three-dimensional packing problem with behavioral constraints, *Comput. Oper. Res.* 28 (2001) 521–535. [https://doi.org/10.1016/S0305-0548\(99\)00132-X](https://doi.org/10.1016/S0305-0548(99)00132-X).
- [75] M. Yao, Z. Chen, L. Luo, R. Wang, H. Wang, Level-set-based partitioning and packing optimization of a printable model, *ACM Trans. Graph.* 34 (2015) 1–11.

<https://doi.org/10.1145/2816795.2818064>.

- [76] Y. Zhao, C.T. Haas, A 3D Irregular Packing Algorithm Using Point Cloud Data, *Comput. Civ. Eng.* 2019. (2019) 201–208. [https://doi.org/ISBN 978-0-7844-1302-9](https://doi.org/ISBN%20978-0-7844-1302-9).
- [77] D.P. Ronconi, A note on constructive heuristics for the flowshop problem with blocking, *Int. J. Prod. Econ.* 87 (2004) 39–48. [https://doi.org/10.1016/S0925-5273\(03\)00065-3](https://doi.org/10.1016/S0925-5273(03)00065-3).
- [78] E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: A survey of the state of the art, *J. Oper. Res. Soc.* 64 (2013) 1695–1724. <https://doi.org/10.1057/jors.2013.71>.
- [79] M.I. Campbell, C.H. Amon, J. Cagan, Optimal three-dimensional placement of heat generating electronic components, *J. Electron. Packag. Trans. ASME.* 119 (1997) 106–113. <https://doi.org/10.1115/1.2792210>.
- [80] Y. Zhao, C. Rausch, C. Haas, Optimizing 3D Irregular Object Packing from 3D Scans Using Metaheuristics, *Adv. Eng. Informatics.* 47 (2021) 101234. <https://doi.org/https://doi.org/10.1016/j.aei.2020.101234>.
- [81] J.E. Lewis, R.K. Ragade, A. Kumar, W.E. Biles, I.T. Ikonen, Using distributed genetic algorithms in three-dimensional bin packing for rapid prototyping machines, *Intell. Syst. Des. Manuf.* 3517 (1998) 45–53. <https://doi.org/10.1117/12.326944>.
- [82] J.E. Lewis, R.K. Ragade, A. Kumar, W.E. Biles, A distributed chromosome genetic algorithm for bin-packing, *Robot. Comput. Integr. Manuf.* 21 (2005) 486–495. <https://doi.org/10.1016/j.rcim.2004.11.017>.
- [83] P. Nebel, G. Richter, K. Weicker, Multi-container loading with non-convex 3D shapes using a GA/TS hybrid, *GECCO'12 - Proc. 14th Int. Conf. Genet. Evol. Comput.* (2012) 1143–1150. <https://doi.org/10.1145/2330163.2330321>.
- [84] J. Zhang, X. Yao, Y. Li, Improved evolutionary algorithm for parallel batch processing machine scheduling in additive manufacturing, *Int. J. Prod. Res.* 0 (2019) 1–20. <https://doi.org/10.1080/00207543.2019.1617447>.
- [85] C. Aladahalli, J. Cagan, K. Shimada, Objective function effect based pattern search - Theoretical framework inspired by 3D component layout, *J. Mech. Des. Trans. ASME.* 129 (2007) 243–254. <https://doi.org/10.1115/1.2406095>.

- [86] C. Aladahalli, J. Cagan, K. Shimada, Objective function effect based pattern search - An implementation for 3D component layout, *J. Mech. Des. Trans. ASME*. 129 (2007) 255–265. <https://doi.org/10.1115/1.2406096>.
- [87] J. Egeblad, Placement of two- and three-dimensional irregular shapes for inertia moment and balance, *Int. Trans. Oper. Res.* 16 (2009) 789–807. <https://doi.org/10.1111/j.1475-3995.2009.00703.x>.
- [88] C. Zhao, L. Jiang, K.L. Teo, K. Lay Teo, A hybrid chaos firefly algorithm for three-dimensional irregular packing problem, *J. Ind. Manag. Optim.* 16 (2020) 409–429. <https://doi.org/10.3934/jimo.2018160>.
- [89] I. Ikonen, W.E. Biles, A. Kumar, R.K. Ragade, J.C. Wissel, A Genetic Algorithm for Packing Three-Dimensional Non-Convex Objects Having Cavities and Holes, *ICGA*. (1997). https://s3.amazonaws.com/academia.edu.documents/39497284/A_Genetic_Algorithm_for_Packing_Three-Di20151028-15046-1htx4.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1516214629&Signature=G14wCBZBVWL2icPrUmaJJxDZ9%2B8%3D&response-content-disposition=inli (accessed January 17, 2018).
- [90] Y. Stoyan, A. Chugay, Packing cylinders and rectangular parallelepipeds with distances between them into a given region, *Eur. J. Oper. Res.* 197 (2009) 446–455. <https://doi.org/10.1016/j.ejor.2008.07.003>.
- [91] Y.E. Stoian, A.M. Chugay, A. V. Pankratov, T.E. Romanova, Two Approaches to Modeling and Solving the Packing Problem for Convex Polytopes, *Cybern. Syst. Anal.* 54 (2018) 585–593. <https://doi.org/10.1007/s10559-018-0059-3>.
- [92] I. Litvinchev, A. Pankratov, T. Romanova, 3D irregular packing in an optimized cuboid container, *IFAC-PapersOnLine*. 52 (2019) 2014–2019. <https://doi.org/10.1016/j.ifacol.2019.11.499>.
- [93] Y.G. Stoyan, A.M. Chugay, Multistage Approach to Solving the Optimization Problem of Packing Nonconvex Polyhedra, *Cybern. Syst. Anal.* 56 (2020). <https://doi.org/10.1007/s10559-020-00241-w>.
- [94] A. Pankratov, T. Romanova, I. Litvinchev, Packing oblique 3D objects, *Mathematics*. 8

- (2020). <https://doi.org/10.3390/math8071130>.
- [95] T. Romanova, Y. Stoyan, A. Pankratov, I. Litvinchev, S. Plankovskyy, Y. Tsegelnyk, O. Shypul, Sparsest balanced packing of irregular 3D objects in a cylindrical container, *Eur. J. Oper. Res.* 291 (2020) 84–100. <https://doi.org/10.1016/j.ejor.2020.09.021>.
- [96] A. Chugay, A. Pankratov, T. Romanova, Irregular layout problem for additive production, *CEUR Workshop Proc.* 2608 (2020) 569–579.
- [97] A.M. Chugay, A. V. Zhuravka, *Packing Optimization Problems and Their Application in 3D Printing*, Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-55506-1_7.
- [98] Y. G. Stoyan, V.N. Patsuk, A method of optimal lattice packing of congruent oriented polygons in the plane, *Eur. J. Oper. Res.* 124 (2000) 204–216.
- [99] J.A. Bennell, J.F. Oliveira, The geometry of nesting problems: A tutorial, *Eur. J. Oper. Res.* 184 (2008) 397–415. <https://doi.org/10.1016/j.ejor.2006.11.038>.
- [100] A.R. Forrest, *COMPUTATIONAL GEOMETRY - ACHIEVEMENTS AND PROBLEMS*, (1974) 17–44. <https://doi.org/10.1016/B978-0-12-079050-0.50007-2>.
- [101] J. Rossignac, Geometric simplification and compression, *Comput. Geom.* (2013) 1–20.
- [102] I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, *5th Int. Conf. Learn. Represent. ICLR 2017 - Work. Track Proc.* (2019) 1–15.
- [103] H. Hu, X. Zhang, X. Yan, L. Wang, Y. Xu, Solving a new 3D bin packing problem with deep reinforcement learning method, *ArXiv.* (2017).
- [104] M. AL-Alawi, Y. Mohamed, A. Bouferguene, Application of industrial pipelines data generator in the experimental analysis: Pipe spooling optimization problem definition, formulation, and testing, *Adv. Eng. Informatics.* 43 (2020) 101007. <https://doi.org/10.1016/j.aei.2019.101007>.
- [105] Y.G. Stoyan, N.I. Gil, A. Pankratov, G. Scheithauer, Packing non-convex polytopes into a parallelepiped, *Math-Nm-06-2004.* (2004). <http://www.math.tu-dresden.de/~scheith/ABSTRACTS/PREPRINTS/04-non-conv.pdf>.

- [106] L. Yang, J.C.P. Cheng, Q. Wang, Semi-automated generation of parametric BIM for steel structures based on terrestrial laser scanning data, *Autom. Constr.* 112 (2020) 103037. <https://doi.org/10.1016/j.autcon.2019.103037>.
- [107] J.-F. Hullo, G. Thibault, C. Boucheny, F. Dory, A. Mas, Multi-Sensor As-Built Models of Complex Industrial Architectures, *Remote Sens.* 7 (2015) 16339–16362. <https://doi.org/10.3390/rs71215827>.
- [108] T. Rabbani, F. Van Den Heuvel, 3D INDUSTRIAL RECONSTRUCTION BY FITTING CSG MODELS TO A COMBINATION OF IMAGES AND POINT CLOUDS, n.d.
- [109] FaroArm® - Portable 3D Measurement Arm for any application, (n.d.). <https://www.faro.com/products/3d-manufacturing/faroarm/#main%3E> (accessed May 8, 2020).
- [110] O. Inc., Structure Sensor - 3D scanning, agostomented reality, and more for mobile devices, (2016). <https://structure.io/> (accessed May 23, 2021).
- [111] MeshLab, (n.d.). <https://www.meshlab.net/> (accessed May 25, 2021).
- [112] M. Tenney, Point Clouds to Mesh in “MeshLab,” CAST Tech. Publ. Ser. Number 100 (2012). <http://gmvc.cast.uark.edu/scanning/point-clouds-to-mesh-in-meshlab/>.
- [113] M. Kazhdan, H. Hoppe, Screened poisson surface reconstruction, *ACM Trans. Graph.* 32 (2013) 1–13. <https://doi.org/10.1145/2487228.2487237>.
- [114] M. Kazhdan, M. Bolitho, H. Hoppe, Poisson Surface Reconstruction, *Eurographics Symp. Geom. Process.* (2006).
- [115] D.P. Bertsekas, A. Com, *Network Optimization: Continuous and Discrete Models* WWW site for book information and orders, MA: Athena Scientific, Belmont, 1998. <http://www.athenasc.com> (accessed October 7, 2020).
- [116] S.P. Boyd, L. Vandenberghe, *Convex optimization* / Stephen Boyd ; Lieven Vandenberghe, (2008). https://books.google.ca/books?hl=en&lr=&id=mYm0bLd3fcoC&oi=fnd&pg=PR11&dq=S.+Boyd+and+L.+Vandenberghe,+Convex+Optimization&ots=tf2TtILzNX&sig=8YgIoBDWdN7_OXQhLIjRjftxA8&redir_esc=y#v=onepage&q=S.+Boyd+and+L.+Vandenberghe%2C+Convex+Optimization&f=false (accessed October 7, 2020).

- [117] K. Miettinen, M.M. Mäkelä, J. Toivanen, Numerical comparison of some penalty-based constraint handling techniques in genetic algorithms, *J. Glob. Optim.* 27 (2003) 427–446. <https://doi.org/10.1023/A:1026065325419>.
- [118] K. Deb, An efficient constraint handling method for genetic algorithms, *Comput. Methods Appl. Mech. Eng.* 186 (2000) 311–338. [https://doi.org/10.1016/S0045-7825\(99\)00389-8](https://doi.org/10.1016/S0045-7825(99)00389-8).
- [119] D. Rutten, Galapagos: On the logic and limitations of generic solvers, *Archit. Des.* 83 (2013) 132–135. <https://doi.org/10.1002/ad.1568>.
- [120] H. Kim, C.T. Haas, A.F. Rauch, C. Browne, 3D image segmentation of aggregates from laser profiling, *Comput. Civ. Infrastruct. Eng.* 18 (2003) 254–263. <https://doi.org/10.1111/1467-8667.00315>.
- [121] H. Kim, C.T. Haas, A.F. Rauch, C. Browne, Dimensional ratios for stone aggregates from three-dimensional laser scans, *J. Comput. Civ. Eng.* 16 (2002) 175–183. [https://doi.org/10.1061/\(ASCE\)0887-3801\(2002\)16:3\(175\)](https://doi.org/10.1061/(ASCE)0887-3801(2002)16:3(175)).
- [122] H. Kim, C.T. Haas, A.F. Rauch, C. Browne, Wavelet-based three-dimensional descriptors of aggregate particles, *Transp. Res. Rec.* 1787 (2002) 109–116. <https://doi.org/10.3141/1787-12>.
- [123] C. Browne, A.F. Rauch, C.T. Haas, H. Kim, Performance Evaluation of Automated Machines for Measuring Gradation of Aggregates, *Geotech. Test. J.* 26 (2003) 373–381. <https://doi.org/10.1520/gtj11254j>.
- [124] H. Kim, A.F. Rauch, C.T. Haas, Automated quality assessment of stone aggregates based on laser imaging and a neural network, *J. Comput. Civ. Eng.* 18 (2004) 58–64. [https://doi.org/10.1061/\(ASCE\)0887-3801\(2004\)18:1\(58\)](https://doi.org/10.1061/(ASCE)0887-3801(2004)18:1(58)).
- [125] A. Pinto, I.L. Nunes, R.A. Ribeiro, Occupational risk assessment in construction industry - Overview and reflection, *Saf. Sci.* 49 (2011) 616–624. <https://doi.org/10.1016/j.ssci.2011.01.003>.
- [126] S. You, J.H. Kim, S.H. Lee, V. Kamat, L.P. Robert, Enhancing perceived safety in human–robot collaborative construction using immersive virtual environments, *Autom. Constr.* 96 (2018) 161–170. <https://doi.org/10.1016/j.autcon.2018.09.008>.
- [127] G. Ballard, G. Howell, Shielding production: Essential step in production control, *J. Constr. Eng. Manag.* 124 (1998) 11–17. [https://doi.org/10.1061/\(ASCE\)0733-9364\(1998\)124:1\(11\)](https://doi.org/10.1061/(ASCE)0733-9364(1998)124:1(11)).

- [128] C. Hendrickson, C.T. Hendrickson, T. Au., Project management for construction: Fundamental concepts for owners, engineers, architects, and builders., 1989.
- [129] H. Abou-Ibrahim, F. Hamzeh, E. Zankoul, S. Munch Lindhard, L. Rizk, Understanding the planner's role in lookahead construction planning, *Prod. Plan. Control.* 30 (2019) 271–284. <https://doi.org/10.1080/09537287.2018.1524163>.
- [130] A. Hertz, M. Widmer, Guidelines for the use of meta-heuristics in combinatorial optimization, *Eur. J. Oper. Res.* 151 (2003) 247–252. [https://doi.org/10.1016/S0377-2217\(02\)00823-8](https://doi.org/10.1016/S0377-2217(02)00823-8).
- [131] J. Du, Z. Zou, Y. Shi, D. Zhao, Zero latency: Real-time synchronization of BIM data in virtual reality for collaborative decision-making, *Autom. Constr.* 85 (2018) 51–64. <https://doi.org/10.1016/j.autcon.2017.10.009>.
- [132] J.J. Lin, M. Golparvar-Fard, Visual and Virtual Production Management System for Proactive Project Controls, *J. Constr. Eng. Manag.* 147 (2021) 04021058. [https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0002045](https://doi.org/10.1061/(ASCE)CO.1943-7862.0002045).
- [133] S. Alizadehsalehi, A. Hadavi, J.C. Huang, From BIM to extended reality in AEC industry, *Autom. Constr.* 116 (2020) 103254. <https://doi.org/10.1016/j.autcon.2020.103254>.
- [134] S. Bükrü, M. Wolf, O. Golovina, J. Teizer, Using Field of View and Eye Tracking for Feedback Generation in an Augmented Virtuality Safety Training, in: *Constr. Res. Congr. 2020 Safety, Work. Educ.*, American Society of Civil Engineers, Reston, VA, 2020: pp. 625–632.
- [135] M. Kassem, L. Benomran, J. Teizer, Virtual environments for safety learning in construction and engineering: seeking evidence and identifying gaps for future research, *Vis. Eng.* 5 (2017) 1–15. <https://doi.org/10.1186/s40327-017-0054-1>.
- [136] A.A. Muhammad, I. Yitmen, S. Alizadehsalehi, T. Celik, Adoption of Virtual Reality (VR) for Site Layout Optimization of Construction Projects, *Tek. Dergi.* (2020) 9833–9850. <https://doi.org/10.18400/tekderg.423448>.
- [137] S. Bhandari, M.R. Hallowell, L. Van Boven, K.M. Welker, M. Golparvar-Fard, J. Gruber, Using Augmented Virtuality to Examine How Emotions Influence Construction-Hazard Identification, Risk Assessment, and Safety Decisions, *J. Constr. Eng. Manag.* 146 (2019) 04019102. [https://doi.org/10.1061/\(ASCE\)CO.1943-7862.0001755](https://doi.org/10.1061/(ASCE)CO.1943-7862.0001755).

- [138] INTERNATIONAL ATOMIC ENERGY AGENCY, Handling and Processing of Radioactive Waste from Nuclear Applications, IAEA, Vienna, 2001.
- [139] Unity Real-Time Development Platform | 3D, 2D VR & AR Engine, (n.d.). <https://unity.com/> (accessed May 23, 2021).
- [140] VIVE Canada | Discover Virtual Reality Beyond Imagination, (n.d.). <https://www.vive.com/ca/> (accessed May 23, 2021).
- [141] MINI TRACTOR | 3D Land | Unity Asset Store, (n.d.). <https://assetstore.unity.com/packages/3d/vehicles/land/mini-tractor-68879> (accessed April 27, 2021).
- [142] GitHub - giacomelli/GeneticSharp, (n.d.). <https://github.com/giacomelli/GeneticSharp> (accessed April 27, 2021).
- [143] S.S. Shapiro, M.B. Wilk, An Analysis of Variance Test for Normality (Complete Samples), *Biometrika*. 52 (1965) 591. <https://doi.org/10.2307/2333709>.
- [144] B.L. Welch, On the Comparison of Several Mean Values: An Alternative Approach, *Biometrika*. 38 (1951) 330. <https://doi.org/10.2307/2332579>.
- [145] P.A. Games, J.F. Howell, Pairwise Multiple Comparison Procedures with Unequal N's and/or Variances: A Monte Carlo Study, *J. Educ. Stat.* 1 (1976) 113–125. <https://doi.org/10.3102/10769986001002113>.
- [146] M. Pytlos, M. Gilbert, C.C. Smith, Modelling granular soil behaviour using a physics engine, *Geotech. Lett.* 5 (2015) 243–249. <https://doi.org/10.1680/jgele.15.00067>.
- [147] L. Al Khateeb, K. Anupam, S. Erkens, T. Scarpas, Micromechanical simulation of porous asphalt mixture compaction using discrete element method (DEM), *Constr. Build. Mater.* 301 (2021) 124305. <https://doi.org/10.1016/j.conbuildmat.2021.124305>.
- [148] M. Fang, D. Park, J.L. Singuranayo, H. Chen, Y. Li, Aggregate gradation theory, design and its impact on asphalt pavement performance: a review, <https://doi.org/10.1080/10298436.2018.1430365>. 20 (2018) 1408–1424. <https://doi.org/10.1080/10298436.2018.1430365>.
- [149] O. Pouliquen, M. Nicolas, P.D. Weidman, Crystallization of non-Brownian Spheres under Horizontal Shaking, *Phys. Rev. Lett.* 79 (1997) 3640–3643.

<https://doi.org/10.1103/PhysRevLett.79.3640>.

- [150] A. Bhandwalidar, M. Ghule, A. Gandhi, N. Bansode, Physics Engine: Deformation Simulation, (n.d.). www.ijcsit.com (accessed December 21, 2021).
- [151] M. Clayton, J. Kunz, M. Fischer, The Charrette Test Method, (1998) 27.
- [152] NewtonVR | Tools | Unity Asset Store, (n.d.).
<https://assetstore.unity.com/packages/tools/newtonvr-75712> (accessed October 21, 2021).
- [153] Non-Convex Mesh Collider. Automatic Generator | Level Design | Unity Asset Store, (n.d.).
<https://assetstore.unity.com/packages/tools/level-design/non-convex-mesh-collider-automatic-generator-117273> (accessed October 21, 2021).
- [154] UnityLodeRunner/ObjImporter.cs at master · phildini/UnityLodeRunner · GitHub, (n.d.).
<https://github.com/phildini/UnityLodeRunner/blob/master/LodeRunner/Assets/AstarPathfindingProject/Generators/Utilities/ObjImporter.cs> (accessed October 21, 2021).
- [155] CSV Reader for unity · GitHub, (n.d.).
<https://gist.github.com/kennir/77a216141ca8fb9efacaf52cebe43307> (accessed October 21, 2021).
- [156] GitHub - giacomelli/GeneticSharp: GeneticSharp is a fast, extensible, multi-platform and multithreading C# Genetic Algorithm library that simplifies the development of applications using Genetic Algorithms (GAs)., (n.d.). <https://github.com/giacomelli/GeneticSharp> (accessed October 21, 2021).

Appendices

Appendix A
Metaheuristic-based Packing Optimization Algorithm

Appendix A shows the details of the metaheuristic-based packing optimization algorithm. The algorithm is implemented in Rhinoceros® using Grasshopper, a visual programming language and environment.

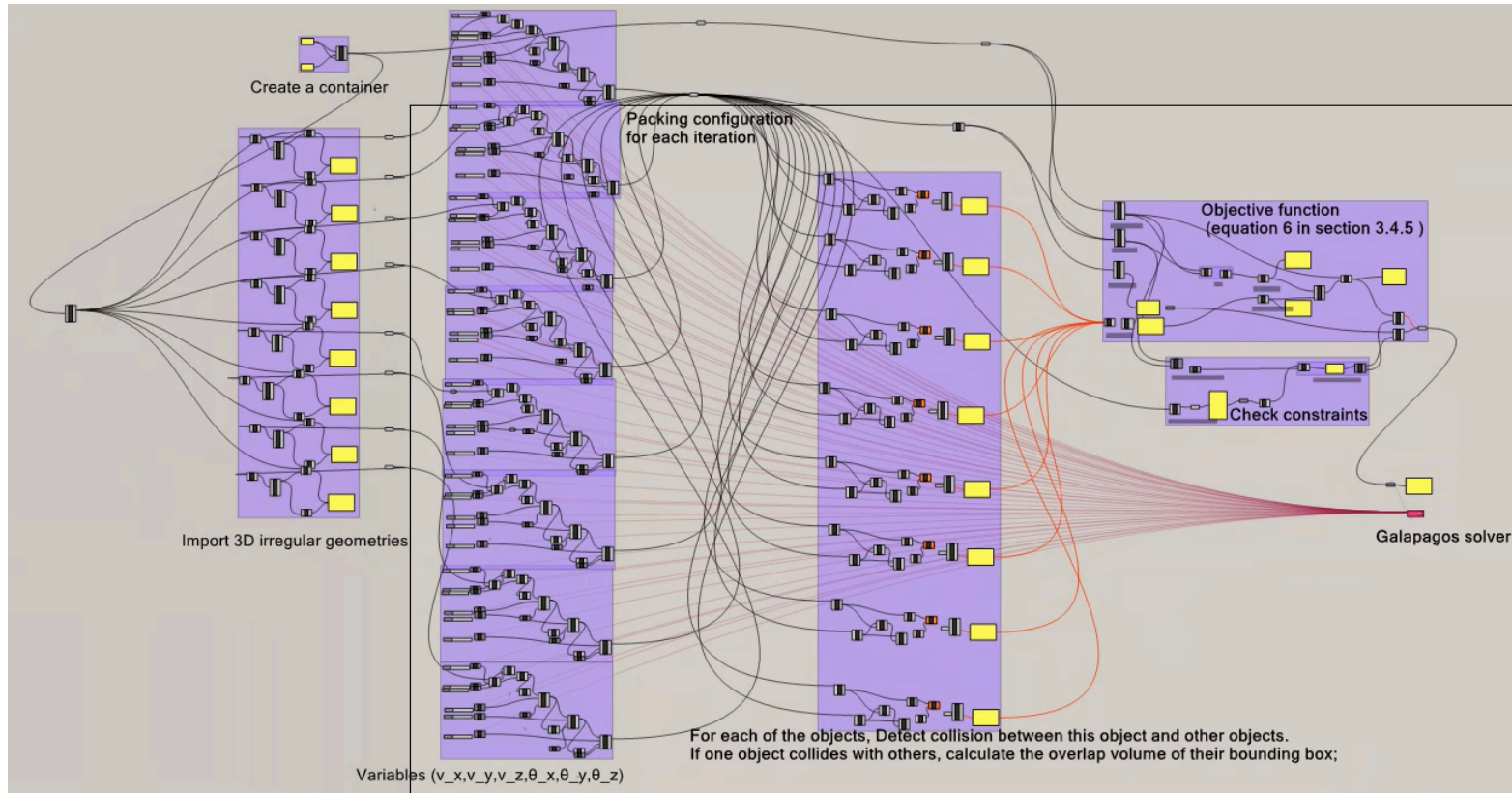
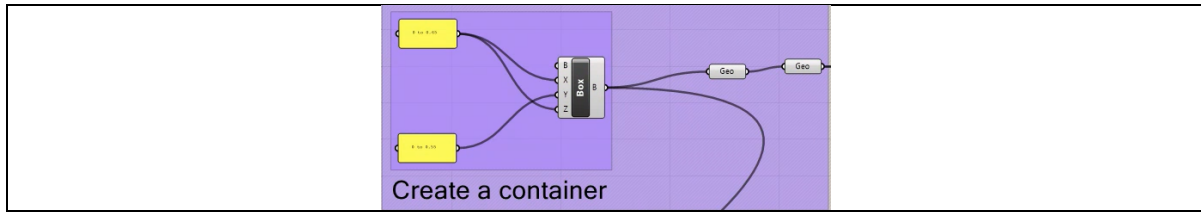
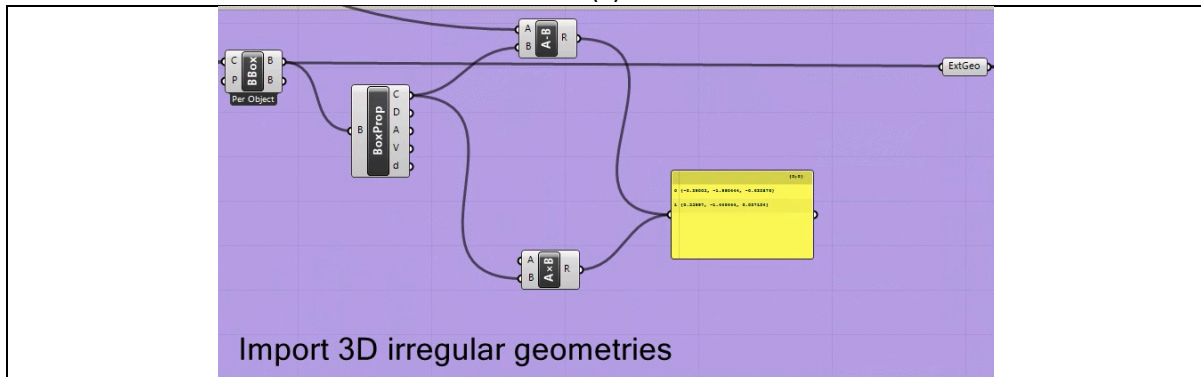


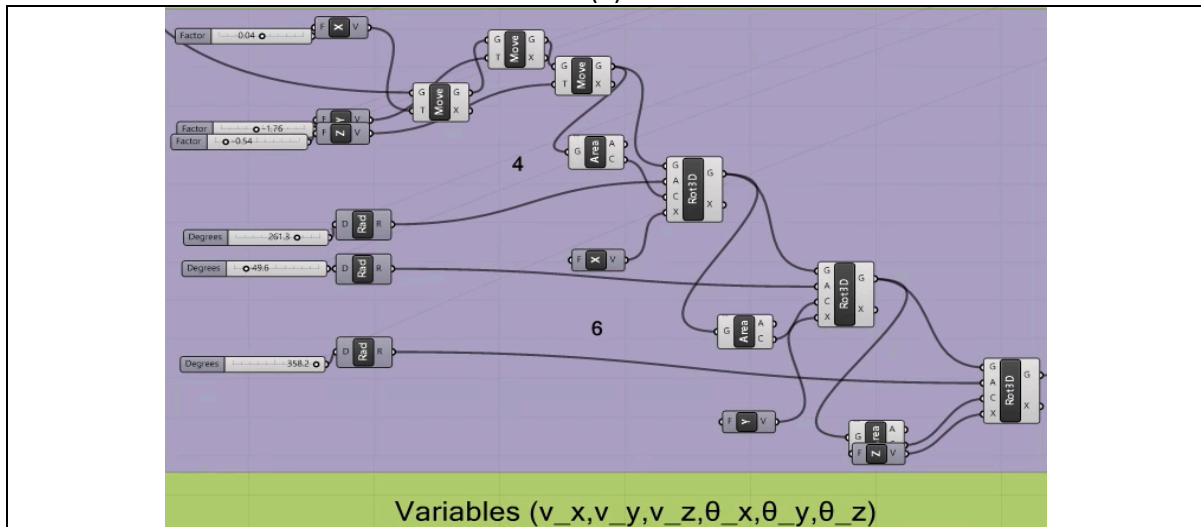
Figure-A-1. Overview of the autonomous packing algorithm



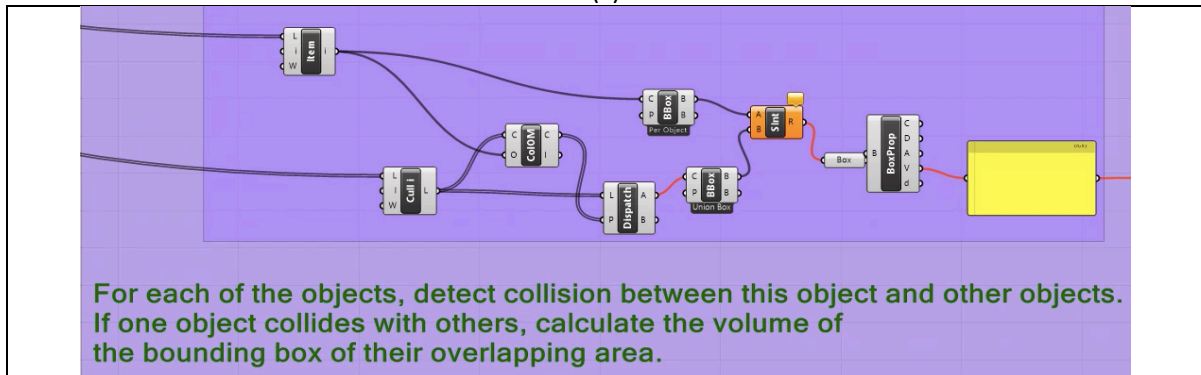
(a)



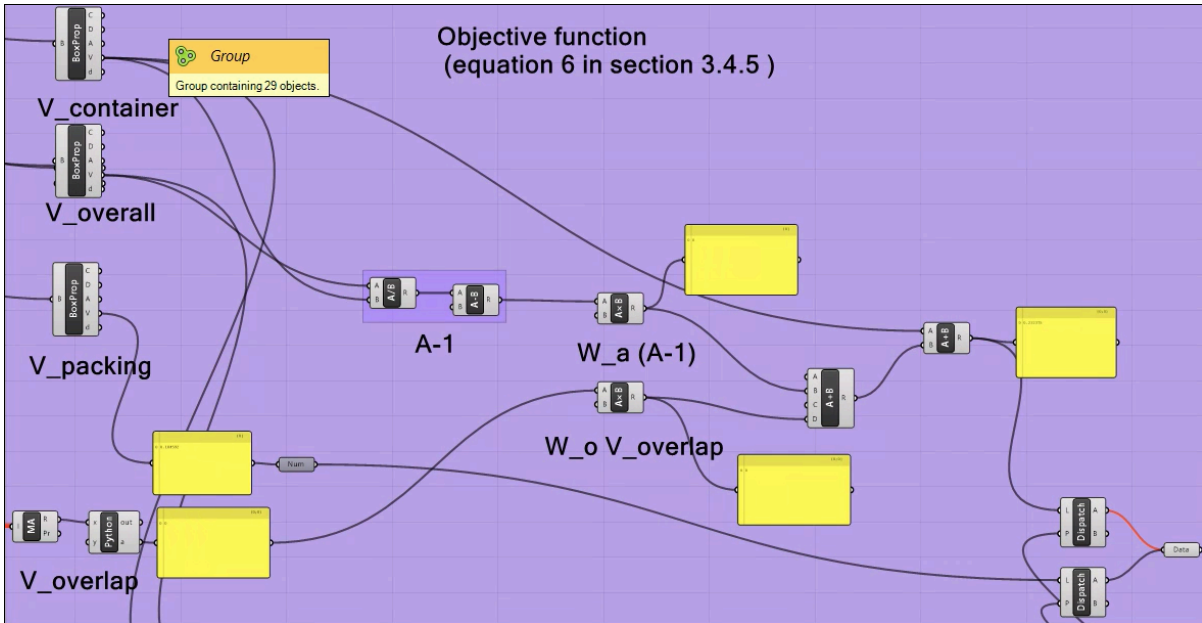
(b)



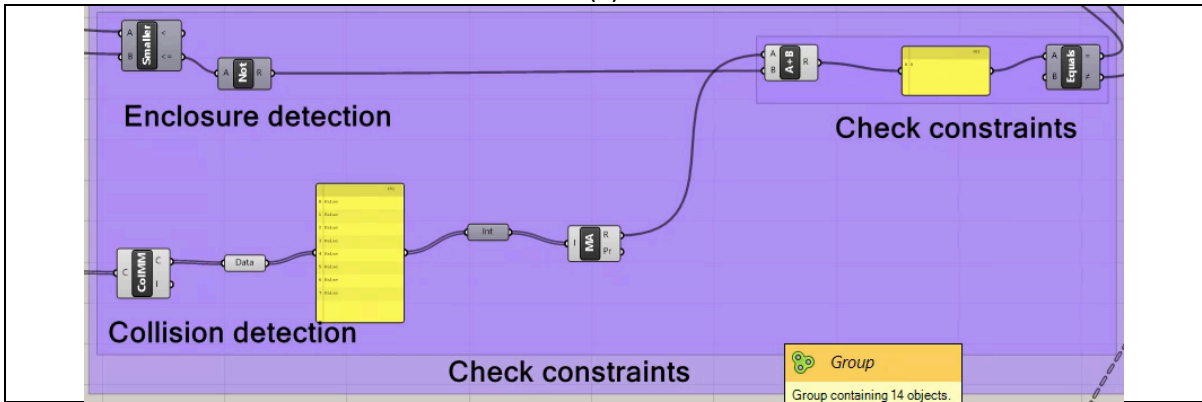
(c)



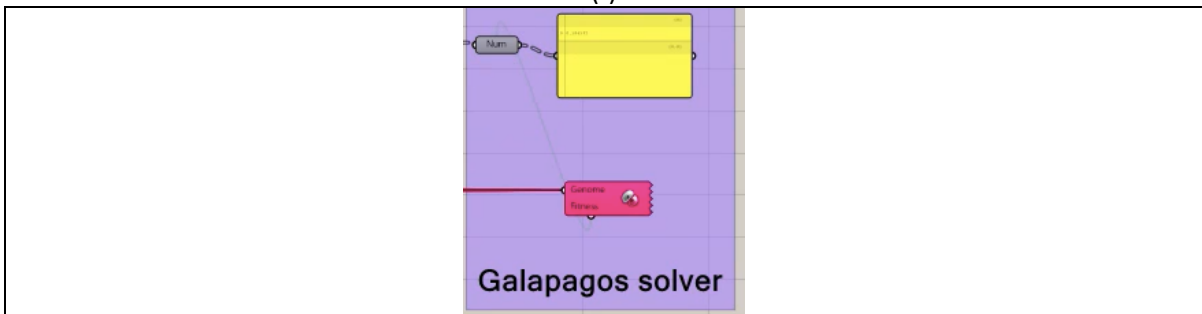
(d)



(e)



(f)



(g)

Figure-A-2. Details of the autonomous packing algorithm.

Appendices B
Sensitivity Analysis

Appendix B shows the results from the sensitivity analysis in Chapter 3.

Table B-1. Packing results of subset 1 with 5 random starting positions

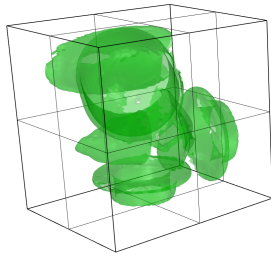
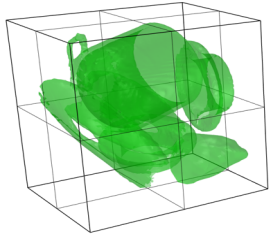
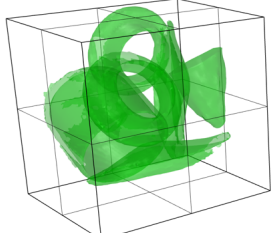
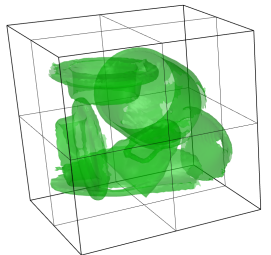
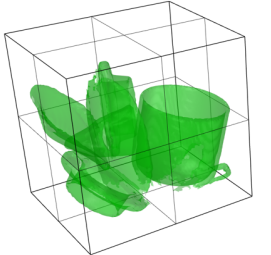
Starting position	Packing configuration	Feasible solution	Outside container %	Overlapping volume	Fitness value	Time (mins)
1		✓	0	0	0.165968	20
2		✓	0	0	0.176289	20
3		✓	0	0	0.163752	20
4		✓	0	0	0.159645	20
5		✓	0	0	0.18837	20

Table B-2. Packing results of subset 2 with 5 random starting positions

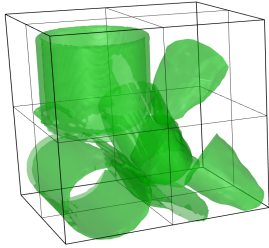
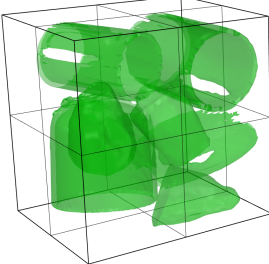
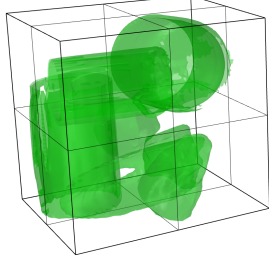
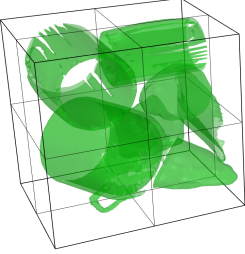
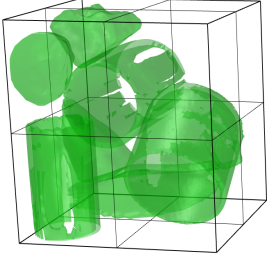
Starting position	Packing configuration	Feasible solution	Outside container %	Overlapping volume	Fitness value	Time (mins)
1		✓	0	0	0.218581	20
2		✓	0	0	0.189769	20
3		✗	0.0426%	0	0.232801	20
4		✓	0	0	0.190859	20
5		✗	0	0.004411	0.276485	20

Table B-3. Packing results of subset 3 with 5 random starting positions

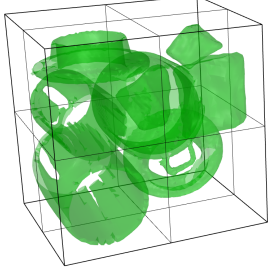
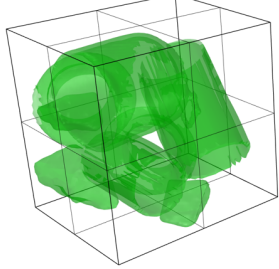
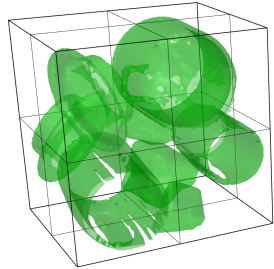
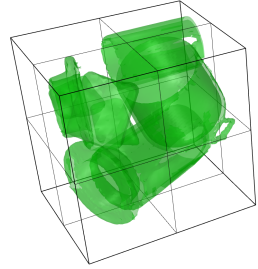
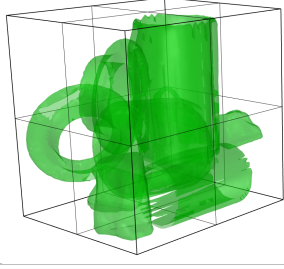
Starting position	Packing configuration	Feasible solution	Outside container %	Overlapping volume	Fitness value	Time (mins)
1		✓	0	0	0.205351	20
2		✗	0	0.000342	0.235795	20
3		✗	1.998%	0	0.252355	20
4		✓	0	0	0.194667	20
5		✗	0	0.001087	0.243245	20

Table B-4. Packing results of subset 4 with 5 random starting positions

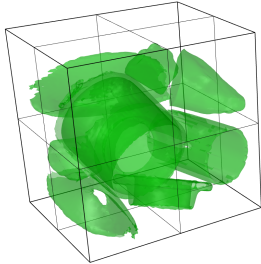
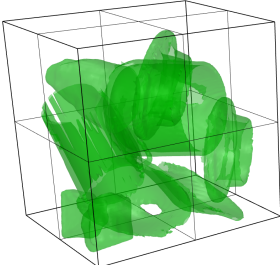
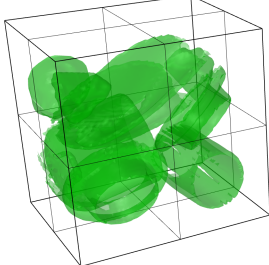
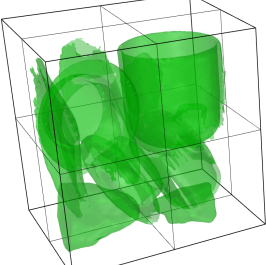
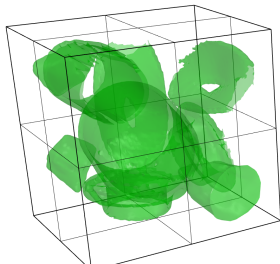
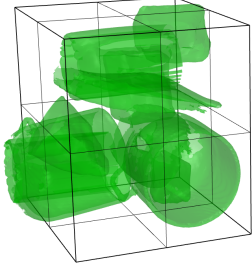
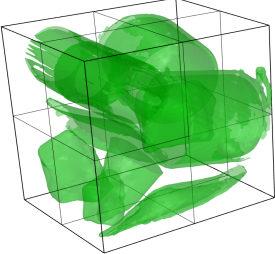
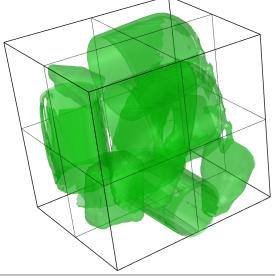
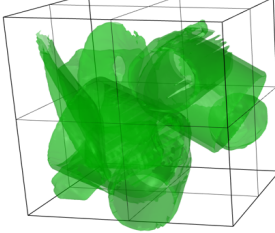
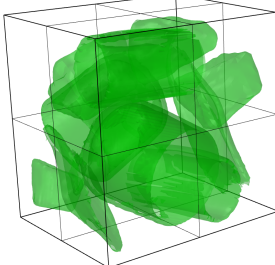
Starting position	Packing configuration	Feasible solution	Outside container %	Overlapping volume	Fitness value	Time (mins)
1		✘	3.7389%	0	0.269764	20
2		✘	0.1717%	0.001543	0.249432	20
3		✔	0	0	0.17428	20
4		✔	0	0	0.163724	20
5		✔	0	0	0.187925	20

Table B-5. Packing results of subset 5 with 5 random starting positions

Starting position	Packing configuration	Feasible solution	Outside container %	Overlapping volume	Fitness value	Time (mins)
1		✘	34.2099%	0.002176	0.596234	20
2		✘	22.0449%	0.000489	0.457714	20
3		✘	8.7931%	0.010859	0.428896	20
4		✘	21.0831%	0	0.443206	20
5		✘	22.3042%	0	0.455417	20

Appendices C
VR Packing Environments

Appendix C includes the details regarding the user interface design of the VR packing environments as well as related algorithms described in Chapter 4.

C.1 User interface (UI) components



Figure-C-1. Screenshot of the UI of the VR packing platform

- Evaluation metrics (criteria, sliders, criteria values)
- Timer: The timer displays and keeps track of packing time
- Start and Stop Buttons
 - The start button initiates the timer and invokes the *SetUp* function in *ConfigReporter* class, which generates packing configuration reports containing information about the ongoing configuration and its criteria value during the packing process. Additionally, it also calls *SetTextOpacity* in Sequential packing scenario to conceal the criteria values until the packing process is complete
 - The stop button stops the timer and calls the *Save* function in *ConfigReporter* class to save the reports. Additionally, it also calls *SetTextOpacity* in Sequential packing scenario to display the criteria values once the packing process is complete
- RunGA Button: RunGA button calls the *RunGA* function in *ObjectManagerGA* class, then it sets up the GA controller to run the GA algorithm.
- Importing Components
 - Number of meshes to import n
 - Decrement button (-) allows you to decrease the number of meshes to be imported, $n \geq 1$.

- Increment button (+) allows you to increase the number of meshes to be imported, $n \leq 15$. The upper bound can be changed.
- Import Meshes button calls the *ImportMeshes* function in *Import* class which randomly selects n number of meshes and imports to Unity
- Import Config button calls the *ImportFromConfig* function in *ImportFromFiles* class which imports packing configurations stored in *Assets/Resources/Config* file

C.2 Assets and libraries:

- NewtonVR [152]: Manages all VR interactions (i.e. dealing with controller inputs and interacting with objects)
- Plawius-NonConvexCollider [153]: Generates multiple convex mesh colliders to form a non-convex mesh collider
- ObjectImporter [154]: Imports mesh data from 3D object files (.obj) and creates a mesh in Unity
- CSVReader [155]: Reads data from CSV files (used for importing from config files)
- GeneticSharp [156]: Library with built-in classes for Genetic Algorithm in C#

C.3 Scripts

C.3.1 Common scripts (used in all four scenarios)

- CollidersCounter: Computes criteria values for objects contained within the container (may include the objects that are slightly outside of the container as it uses the centre of the objects to determine if the object is in the container or not). See Algorithm C-1.
- ConfigReporter: Creates packing configuration reports. See Algorithm C-2.
- Import: Randomly import n number of objects where n is defined by *NumMeshes* field on the Packing UI. See Algorithm C-3.
- ImportFromFiles: Imports packing configurations stored in *Assets/Resources/Config* file. See Algorithm C-4.
- Property: Sets up properties (weight, radiation, and volume) of each object. See Algorithm C-5.
- PopupWindow: Displays warning messages on the Packing UI when the weight and/or radiation limits are reached. See Algorithm C-6.

- ShowValueScript: Displays the criteria values of the current packing configuration in percentage on the Packing UI. See Algorithm C-7.
- Timer: Text on Packing UI to keep track of time. See Algorithm C-8.
- SlidersManager (See Algorithm C-9)
 - Visualizes the criteria values displayed by the sliders on the Packing UI
 - Updates the packing configuration report
 - Depending on the scenario, hides or displays the evaluation metrics. Hides the Packing UI if running *Sequential Packing* or *Excavator Packing* and only show the evaluation metrics when the packing is done. Displays the Packing UI if running *Autonomous Packing* or *Trial-and-Error*. The evaluation metrics are updated to give real-time feedback to the user

C.3.2 Scripts used in the autonomous packing scenario

- ObjectManagerGA: Reads in objects that is selected for autonomous packing algorithm and starts GA process. See Algorithm C-10.
- GAController (See Algorithm C-11)
 - Initializes necessary GA components (i.e., Population, Selection, Crossover, Mutation, Reinsertion)
 - Evolves generations of GA
 - Reports the best chromosome after each generation performs
- PackingChromosome: Chromosome class inheriting from *ICromosome* in *GeneticSharp* library. It implements functions to create, clone, modify, and evaluate chromosomes. See Algorithm C-12.
 - Genes are 4×1 vectors. the first item represents the ID of the object, and the follow rows form a binary string which corresponds to the orientation of the object for autonomous packing
 - Chromosome is a $4 \times k$ matrix where n represents the number of selected objects for GA. There are k number of genes in the chromosome.
- PackingOrderBasedCrossover: Order based crossover modified from GeneticSharp. For GA autonomous packing algorithm integrated in the VR packing platform, since genes contains orientation information, the offsprings mix this information from both parents. For example, when the offspring copies genes from the first parent (1 2 3 4 5 6 7 8) to form (1 2 3 * * * 7

8), it copies the first parent's orientation information. When fills up the missing spots with the second parent, the offsprings gets orientation information from the second parent. See Algorithm C-13.

- PackingFlipBitMutation: Modified flip bit mutation from GeneticSharp. Randomly select an object to mutate its orientation. Randomly chooses the $j \in (1,2,3)$ number of bits to flip. See Algorithm C-14.
- SelectionManager: Handles the selection/deselection of objects. When the object is selected, it highlights the object with yellow and stores that object in selections list. When the object is deselected, it changes the object material to the original colour and removes the object from the selections list. The final selected objects will be packed autonomously using GA. See Algorithm C-15.
- BLFPackingInwards: Runs BLF placement heuristic and returns the packing efficiency and the total weight and radiation percentage of the objects in the container. See Algorithm C-16.

Algorithm C-1. CollidersCounter

```
using UnityEngine;
using System.Collections.Generic;
using System;
using System.Collections;

public class CollidersCounter : MonoBehaviour
{
    [Range(0.0f, 10.0f)]
    public float multiplierX; //resize the container in the beginning
    [Range(0.0f, 10.0f)]
    public float multiplierY;
    [Range(0.0f, 10.0f)]
    public float multiplierZ;

    //Make sure to assign this in the Inspector window
    List<GameObject> Wastes = new List<GameObject>();
    List<GameObject> CurrentlyPackedWastes = new List<GameObject>();
    List<Property> Property_script= new List<Property>();
    BoxCollider m_Collider;
    public float Totalvolume;
    public float Totalradioactivity;
    public float Totalweight;
    public float weightLimitation = 367f;
    public float doseLimitation = 1.6f;
    public Vector3 centerOfGravity= new Vector3(0.0f, 0.0f, 0.0f);
```

```

public int count;
int[] collidObjects=new int[100];
Vector3 sumMassTimeslocation= new Vector3(0.0f, 0.0f, 0.0f);
public float colliderVolume;

void Awake()
{
    Vector3 multiplier = new Vector3(multiplierX, multiplierY, multiplierZ);

    if (multiplier != null && multiplier.x > 0 && multiplier.y > 0 && multiplier.z > 0)
    {
        Vector3 initialScale = transform.localScale;

        transform.localScale = Vector3.Scale(transform.localScale, multiplier);

        Vector3 differenceInSize = transform.localScale - initialScale;
        differenceInSize = differenceInSize / 2; //get half of the size increase (to move the box that
much more)

        //Move the box more outwards (also away from the ground)
        transform.position = new Vector3(transform.position.x + differenceInSize.x,
            transform.position.y + differenceInSize.y,
            transform.position.z - differenceInSize.z);
    }
}

void Start()
{
    //Fetch the Collider from the GameObject this script is attached to
    m_Collider = GetComponent<BoxCollider>();
    colliderVolume = m_Collider.size[0] * m_Collider.size[1] * m_Collider.size[2] ;
    colliderVolume = colliderVolume * gameObject.transform.localScale.x *
gameObject.transform.localScale.y * gameObject.transform.localScale.z;
    ResetWastes();
}

void Update()
{
    CalculateTotalProperties();
}

public void CalculateTotalProperties()
{
    count = 0;
    Totalvolume = 0;
    Totalradioactivity = 0;
    Totalweight = 0;
    for (int i = 0; i < Wastes.Count; i++)

```

```

    {
        if (m_Collider.bounds.Contains(Wastes[i].transform.position))
        {
            Totalvolume = Totalvolume + Wastes[i].GetComponent<Property>().GetVolume();
            Totalradioactivity = Totalradioactivity +
Wastes[i].GetComponent<Property>().GetRadioactivity();
            Totalweight = Totalweight + Wastes[i].GetComponent<Property>().GetWeight();
            collidObjects[count] = i;
            count = count + 1;
        }
    }
    sumMassTimeslocation = new Vector3(0,0,0);

    if (count > 0)
    {
        for (int i = 0; i < count; i++)
        {
            sumMassTimeslocation = sumMassTimeslocation +
Wastes[collidObjects[i]].GetComponent<Property>().GetVolume() *
Wastes[collidObjects[i]].GetComponent<Property>().centerOfGravity;
            //Debug.Log("sumMassTimeslocation: " + sumMassTimeslocation);
        }

        centerOfGravity = sumMassTimeslocation / Totalvolume;
    }

    else
        centerOfGravity = Vector3.zero;
}

public List<GameObject> GetWastes()
{
    return Wastes;
}

public void ResetWastes()
{
    Wastes = new List<GameObject>(GameObject.FindGameObjectsWithTag("Interactable"));
}

public List<GameObject> GetCurrentlyPackedWastes()
{
    CurrentlyPackedWastes = new List<GameObject>();

    for (int i = 0; i < Wastes.Count; i++)
    {
        if (m_Collider.bounds.Contains(Wastes[i].transform.position))
        {

```



```

        CurrentlyPackedWastes.Add(Wastes[i]);
    }
}

//Debug.Log("Currently packing: " + string.Join(", ", CurrentlyPackedWastes));

return CurrentlyPackedWastes;
}
}

```

Algorithm C-2. -ConfigReporter

```

using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System;
using UnityEngine.SceneManagement;

public class ConfigReporter : MonoBehaviour
{
    private List<string[]> configRowData = new List<string[]>();
    private List<string[]> reportRowData = new List<string[]>();
    private List<string> selections = new List<string>();

    GameObject[] objects;
    BoxCollider counter_collider;
    ObjectManager manager;

    //private configOutputStream;
    //private reportOutputStream;

    private string configFilepath;
    private string reportFilepath;

    private bool started = false;
    private bool stopped = false;

    public bool runGA = false;

    void Start()
    {
        configFilepath = GetConfigPath();

        reportFilepath = GetReportPath();
    }
}

```

```

}

public void SetUp()
{
    objects = GameObject.FindGameObjectsWithTag("Interactable");
    counter_collider = GameObject.Find("Counter").GetComponent<BoxCollider>();

    if (SceneManager.GetActiveScene().name == "Autonomous Packing" && runGA)
    {
        var temp =
GameObject.Find("Controller").GetComponent<ObjectManagerGA>().GetSelections();
        foreach (string name in temp)
        {
            selections.Add(name);
        }
    }

    string[] reportHeaders = new string[6] {
        "Packing efficiency",
        "COG",
        "Time",
        "Radiation limit exceed",
        "Weight limit exceed",
        "Number of packed objects"
    };
    reportRowData.Add(reportHeaders);
    started = true;
}

public void AppendToReport(string[] line)
{
    if (started && !stopped)
    {
        reportRowData.Add(line);
    }
}

public void Save()
{
    if (started && !stopped)
    {
        string delimiter = ",";
        // Creating First row of titles manually.
        string[] rowDataTemp = new string[9] {
            "Object",
            "Pos x",

```

```

    "Pos y",
    "Pos z",
    "Rot x",
    "Rot y",
    "Rot z",
    "Packed",
    "Used for GA"
};
configRowData.Add(rowDataTemp);
foreach (GameObject obj in objects)
{
    rowDataTemp = new string[9];
    rowDataTemp[0] = obj.name;
    rowDataTemp[1] = obj.transform.position.x.ToString();
    rowDataTemp[2] = obj.transform.position.y.ToString();
    rowDataTemp[3] = obj.transform.position.z.ToString();
    rowDataTemp[4] = obj.transform.rotation.eulerAngles.x.ToString();
    rowDataTemp[5] = obj.transform.rotation.eulerAngles.y.ToString();
    rowDataTemp[6] = obj.transform.rotation.eulerAngles.z.ToString();
    if (counter_collider.bounds.Contains(obj.transform.position))
    {
        rowDataTemp[7] = "Y";
    }
    else
    {
        rowDataTemp[7] = "N";
    }

    if(SceneManager.GetActiveScene().name == "Autonomous Packing")
    {
        if (selections.Contains(obj.name))
        {
            rowDataTemp[8] = "Y";
        }
        else
        {
            rowDataTemp[8] = "N";
        }
    }
    configRowData.Add(rowDataTemp);

    string[][] configOutput = new string[configRowData.Count][];

    for (int i = 0; i < configOutput.Length; i++)
    {
        configOutput[i] = configRowData[i];
    }
}

```

```

StreamWriter configOutputStream = System.IO.File.CreateText(configFilePath);
int configLength = configOutput.GetLength(0);

StringBuilder configSb = new StringBuilder();

for (int index = 0; index < configLength; index++)
    configSb.AppendLine(string.Join(delimiter, configOutput[index]));

configOutputStream.WriteLine(configSb);
configOutputStream.Close();
}

StreamWriter reportOutputStream = System.IO.File.CreateText(reportFilePath);

string[][] reportOutput = new string[reportRowData.Count][];

for (int i = 0; i < reportOutput.Length; i++)
{
    reportOutput[i] = reportRowData[i];
}

int reportLength = reportOutput.GetLength(0);

StringBuilder reportSb = new StringBuilder();

for (int index = 0; index < reportLength; index++)
    reportSb.AppendLine(string.Join(delimiter, reportOutput[index]));

reportOutputStream.WriteLine(reportSb);
reportOutputStream.Close();
}

stopped = true;
}

// Following method is used to retrieve the relative path as device platform
private string GetConfigPath()
{
    return Application.dataPath + "../Config/Config_" +
System.DateTime.UtcNow.ToString("yyyy-MM-dd-HH-mm-ss") + ".csv";
}

private string GetReportPath()
{
    return Application.dataPath + "../Report/Report_" +
System.DateTime.UtcNow.ToString("yyyy-MM-dd-HH-mm-ss") + ".csv";
}
}

```

Algorithm C-3. Import

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.IO;
using BzKovSoft.ObjectSlicer.Samples;
using NewtonVR;
using Plawius.NonConvexCollider;
using UnityEngine.UI;

public class Import : MonoBehaviour
{
    public Material default_mat;
    private int counter = 0;
    private FileInfo[] files;
    public GameObject parentObject;
    private int num_meshes = 3; //number of meshes imported
    private int max_num_meshes;
    public Timer t;
    private int[] imported_meshes;
    public GameObject num_meshes_field;
    public GameObject import_button;
    private Text num_meshes_text;
    public GameObject decrement_button;
    public GameObject increment_button;
    private Vector3 play_area_pos;
    // public GameObject import_config_button;
    public CollidersCounter colliders_counter;

    void Start()
    {
        string myPath = "C:/Projects/meshes/";
        DirectoryInfo dir = new DirectoryInfo(myPath);
        files = dir.GetFiles("*.obj");
        Debug.Log("There are " + files.Length + " meshes in the folder.");
        max_num_meshes = files.Length;
        imported_meshes = new int[max_num_meshes];
        for (int i = 0; i < max_num_meshes; ++i)
        {
            imported_meshes[i] = 0;
        }
        num_meshes_text = GameObject.Find("NumMeshes").GetComponentInChildren<Text>();
        play_area_pos = GameObject.Find("PlayArea").transform.position;
    }

    public void ImportNumMeshes(int num)
```

```

{
    if (num > max_num_meshes)
    {
        Debug.LogError("Not enough meshes in the folder.");
        return;
    }
    else if (max_num_meshes < 1)
    {
        Debug.LogError("No meshes in the folder.");
        return;
    }

    while (counter < num)
    {
        Debug.Log("Importing mesh #" + (counter + 1));
        int index = UnityEngine.Random.Range(0, max_num_meshes - 1);
        if (imported_meshes[index] == 0)
        {
            CreateMesh(index);
            counter++;
            imported_meshes[index] = 1;
        }
    }
    import_button.SetActive(false);
    num_meshes_field.SetActive(false);
    increment_button.SetActive(false);
    decrement_button.SetActive(false);
    // import_config_button.SetActive(false);
    t.SetUpTimer(num);
    colliders_counter.ResetWastes();
}

public void ImportMeshes()
{
    if (num_meshes > max_num_meshes)
    {
        Debug.LogError("Not enough meshes in the folder.");
        return;
    }
    else if (max_num_meshes < 1)
    {
        Debug.LogError("No meshes in the folder.");
        return;
    }

    while(counter < num_meshes)
    {
        Debug.Log("Importing mesh #" + (counter + 1));
        int index = UnityEngine.Random.Range(0, max_num_meshes - 1);

```

```

        if (imported_meshes[index] == 0)
        {
            CreateMesh(index);
            counter++;
            imported_meshes[index] = 1;
        }
    }
    import_button.SetActive(false);
    num_meshes_field.SetActive(false);
    increment_button.SetActive(false);
    decrement_button.SetActive(false);
    // import_config_button.SetActive(false);
    t.SetupTimer(num_meshes);
    colliders_counter.ResetWastes();
}

public void CreateMesh(int index)
{
    GameObject new_mesh = new GameObject(files[index].Name);
    new_mesh.transform.parent = parentObject.transform;
    var rb = new_mesh.AddComponent<Rigidbody>();
    rb.collisionDetectionMode = CollisionDetectionMode.Continuous;

    var mf = new_mesh.AddComponent<MeshFilter>();
    var renderer = new_mesh.AddComponent<MeshRenderer>();
    var oss = new_mesh.AddComponent<ObjectSlicerSample>();
    var property = new_mesh.AddComponent<Property>();
    var interactable_item = new_mesh.AddComponent<NVRInteractableItem>();

    ObjImporter = new ObjImporter();
    mf.mesh = objImporter.ImportFile(files[index].FullName);
    renderer.material = default_mat;
    oss.defaultSliceMaterial = default_mat;
    property.Setup();
    AddNonConvexColliders(mf.mesh, new_mesh);

    Vector3 random_pos = new Vector3(UnityEngine.Random.Range(play_area_pos.x - 1.0f,
    play_area_pos.x + 1.0f), play_area_pos.y, UnityEngine.Random.Range(play_area_pos.z - 1.0f,
    play_area_pos.z + 1.0f));
    new_mesh.transform.position = random_pos;

    new_mesh.tag = "Interactable";

    GameObject holder = GameObject.Find("Imported Objects Holder");
    if (holder != null) new_mesh.transform.SetParent(holder.transform);
}

public void DecrementNumMeshes()

```

```

{
    if (num_meshes == 1)
    {
        Debug.Log("Can't import less than 1 mesh!");
    }
    else
    {
        num_meshes--;
    }
    num_meshes_text.text = num_meshes.ToString();
}
public void IncrementNumMeshes()
{
    if (num_meshes == max_num_meshes)
    {
        Debug.Log("Can't import more meshes than the max!");
    }
    else
    {
        num_meshes++;
    }
    num_meshes_text.text = num_meshes.ToString();
}

void AddNonConvexColliders(Mesh m, GameObject selected_object)
{
    Mesh[] meshes;
    meshes = API.GenerateConvexMeshes(m, Parameters.Default());

    var collider_asset = NonConvexColliderAsset.CreateAsset(meshes);
    var non_convex = selected_object.AddComponent<NonConvexColliderComponent>();
    non_convex.SetPhysicsCollider(collider_asset);
}
}

```

Algorithm C-4. ImportFromFiles

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.IO;
using BzKovSoft.ObjectSlicer.Samples;
using Plawius.NonConvexCollider;
using UnityEngine.UI;
using NewtonVR;

public class ImportFromFiles : MonoBehaviour {

```



```

public Material default_mat;
public GameObject parentObject;
private int num_meshes = 15;
public Timer t;
public GameObject num_meshes_field;
public GameObject import_button;
public GameObject decrement_button;
public GameObject increment_button;
private Vector3 play_area_pos;
public GameObject import_config_button;
public CollidersCounter colliders_couter;

private List<string> filePaths = new List<string>();
private List<Vector3> objPoses = new List<Vector3>();
private List<Quaternion> objRots = new List<Quaternion>();

void Start ()
{
    play_area_pos = GameObject.Find("PlayArea").transform.position;
    var path = "C:/projects/meshes/";
    var dataset = Resources.Load<TextAsset>("Config");
    string[,] data = CSVReader.SplitCsvGrid(dataset.text);
    Debug.Log(data.GetLength(1));
    for (int i = 1; i < data.GetLength(1); i++)
    {
        if (data[0, i] != null)
        {
            if (System.IO.File.Exists(path + data[0, i]))
            {
                filePaths.Add(path + data[0, i]);
                objPoses.Add(new Vector3(float.Parse(data[1,
i]),float.Parse(data[2, i]),float.Parse(data[3, i])));
                objRots.Add(Quaternion.Euler(float.Parse(data[4,
i]),float.Parse(data[5, i]),float.Parse(data[6, i])));
                Debug.Log(path + data[0, i]);
                Debug.Log(data[1, i] + ", " + data[2, i] + ", " + data[3, i] + ",
" +data[4, i] + ", " + data[5, i] + ", " +data[6, i]);
            }
        }
    }
}

public void ImportFromConfig()
{
    for(int i = 0; i < filePaths.Count; i++) {
        string path = filePaths[i];
        FileInfo f = new FileInfo(path);

```

```

        GameObject new_mesh = new GameObject(f.Name);
        new_mesh.transform.parent = parentObject.transform;
        var rb = new_mesh.AddComponent<Rigidbody>();
        rb.collisionDetectionMode = CollisionDetectionMode.Continuous;

        var mf = new_mesh.AddComponent<MeshFilter>();
        var renderer = new_mesh.AddComponent<MeshRenderer>();
        var oss = new_mesh.AddComponent<ObjectSlicerSample>();
        var property = new_mesh.AddComponent<Property>();
    var interactable_item = new_mesh.AddComponent<NVRInteractableItem>();

    ObjImporter = new ObjImporter();
        mf.mesh = objImporter.ImportFile(f.FullName);
        renderer.material = default_mat;
        oss.defaultSliceMaterial = default_mat;
        property.Setup();
        AddNonConvexColliders(mf.mesh, new_mesh);

        Vector3 random_pos = new
Vector3(UnityEngine.Random.Range(play_area_pos.x-1.0f, play_area_pos.x+1.0f), play_area_pos.y,
UnityEngine.Random.Range(play_area_pos.z - 1.0f, play_area_pos.z + 1.0f));
        new_mesh.transform.position = random_pos;
        new_mesh.tag = "Interactable";
        new_mesh.transform.position = objPoses[i];
        new_mesh.transform.rotation = objRots[i];
    }
    num_meshes = GameObject.FindGameObjectsWithTag("Interactable").Length;
    import_button.SetActive(false);
    num_meshes_field.SetActive(false);
    increment_button.SetActive(false);
    decrement_button.SetActive(false);
    import_config_button.SetActive(false);
    t.SetupTimer(num_meshes);
    colliders_couter.ResetWastes();
}

void AddNonConvexColliders(Mesh m, GameObject selected_object)
{
    Mesh[] meshes;
    meshes = API.GenerateConvexMeshes(m, Parameters.Default());

    var collider_asset = NonConvexColliderAsset.CreateAsset(meshes);
    var non_convex =
selected_object.AddComponent<NonConvexColliderComponent>();
    non_convex.SetPhysicsCollider(collider_asset);
}
}

```

Algorithm C-5. Property

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Property : MonoBehaviour
{
    // Start is called before the first frame update
    public float _obj_weight = 0.0f;
    public float _obj_volume = 0.0f;
    public float _radioactivity = 0.0f;
    //bool nonMeshVolumeCalculate;
    //NonConvexMeshCollider;
    //float volume;
    private Vector3 localCenterOfGravity;
    private Vector3 centerofgravitytotal = new Vector3(0.0f, 0.0f, 0.0f);
    public Vector3 centerOfGravity;
    public int lowOrHighRadiation = 0;

    private Mesh _mesh;
    public void SetUp()
    {
        _mesh = GetComponent<MeshFilter>().sharedMesh;
        _obj_volume = VolumeOfMesh(_mesh);
        _obj_weight = WeightCalculation(_obj_volume);
        _radioactivity = RadioactivityCalculation(_obj_volume);

        string msg = "object: " + this.name + "volume: " + _obj_volume + "\n weight: " + _obj_weight +
            "\n radioactivity: " + _radioactivity;
        //Debug.Log(msg);

        //Debug.Log(this.name + "'s mesh has " + _mesh.triangles.Length + " triangles and " +
            _obj_volume + " volume.");
    }

    public float WeightCalculation(float volume)
    {
        float weight = 0.0f;
        if (gameObject.name.Contains("pipe"))
        {
            weight = 7850 * volume;
        }
        else
        {
            weight = 2400 * volume;
        }
    }
}
```

```

    return weight;
}

public float RadioactivityCalculation(float volume)
{
    float radioactivity = 0f;
    if (gameObject.name.Contains("pipe"))
    {
        radioactivity = 24f * volume;
    }
    else
        radioactivity = 12f * volume;
    return radioactivity;
}

float SignedVolumeOfTriangle(Vector3 p1, Vector3 p2, Vector3 p3)
{
    float v321 = p3.x * p2.y * p1.z;
    float v231 = p2.x * p3.y * p1.z;
    float v312 = p3.x * p1.y * p2.z;
    float v132 = p1.x * p3.y * p2.z;
    float v213 = p2.x * p1.y * p3.z;
    float v123 = p1.x * p2.y * p3.z;
    return (1.0f / 6.0f) * (-v321 + v231 + v312 - v132 - v213 + v123);
}

float VolumeOfMesh(Mesh m)
{
    float v = 0;
    Vector3[] vertices = m.vertices;
    int[] triangles = m.triangles;
    for (int i = 0; i < m.triangles.Length; i += 3)
    {
        Vector3 p1 = vertices[triangles[i + 0]];
        Vector3 p2 = vertices[triangles[i + 1]];
        Vector3 p3 = vertices[triangles[i + 2]];
        v += SignedVolumeOfTriangle(p1, p2, p3);
    }
    v = v * gameObject.transform.localScale.x * gameObject.transform.localScale.y *
    gameObject.transform.localScale.z;

    return Mathf.Abs(v);
}

void Start()
{
    SetUp();
}

```

```

void Update()
{
    centerOfGravity = gameObject.transform.position;
}

public float GetVolume()
{
    return _obj_volume;
}

public float GetWeight()
{
    return _obj_weight;
}

public float GetRadioactivity()
{
    return _radioactivity;
}

public void RecalculateProperties()
{
    _mesh = GetComponent<MeshFilter>().sharedMesh;
    _obj_volume = VolumeOfMesh(_mesh);
    _obj_weight = WeightCalculation(_obj_volume);
    _radioactivity = RadioactivityCalculation(_obj_volume);
}
}

```

Algorithm C-6. PopupWindow

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class PopupWindow : MonoBehaviour
{
    GameObject window;
    public Text messageField;
    Slider weightslider;
    Slider radioactivityslider;
    string message;
    SlidersManager slider_manager;
    string mode = "linear";

    // Show(string)

```

//Display the indicated message in a pop-up window

```
void Start()
{
    if (SceneManager.GetActiveScene().name == "Trial-and-Error" ||
        SceneManager.GetActiveScene().name == "Autonomous Packing")
    {
        mode = "trial";
    }
    if (mode == "trial")
    {
        weightslider = GameObject.Find("WeightSlider").GetComponent<Slider>();
        radioactivityslider = GameObject.Find("RadiationSlider").GetComponent<Slider>();
    }
    else if (mode == "linear")
    {
        slider_manager = GameObject.Find("ScoreUI").GetComponent<SlidersManager>();
    }

    window = GameObject.Find("PopUpWindow");
}

void Update()
{
    if (mode == "trial")
    {
        if (weightslider.value > 1)
        {
            if (radioactivityslider.value <= 1)
            {
                Show("Weight limit exceeded!");
            }

            else
            {
                Show("Weight and radiation limits exceeded!");
            }
        }
    }
    else
    {
        if (radioactivityslider.value > 1)
        {
            Show("Radiation limit exceeded!");
        }
        else
            window.SetActive(false);
    }
}
```

```

    }
    else if (mode == "linear")
    {
        if (slider_manager.Weight > 1)
        {
            if (slider_manager.Radiation <= 1)
            {
                Show("Weight limit exceeded!");
            }
            else
            {
                Show("Weight and radiation limits exceeded!");
            }
        }
    }
    else
    {
        if (slider_manager.Radiation > 1)
        {
            Show("Radiation limit exceeded!");
        }
        else
            window.SetActive(false);
    }
}

}

public void Show(string message)
{
    //messageField.text = message;
    window.SetActive(true);
}

}

```

Algorithm C-7. ShowValueScript

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ShowValueScript : MonoBehaviour
{
    Text percentageText;

    // Start is called before the first frame update
    void Start()
    {

```

```

    percentageText = GetComponent<Text>();
}
// Update is called once per frame
public void textUpdate(float value)
{
    percentageText.text = (value * 100).ToString("n2") + "%";
}
}

```

Algorithm C-8. Timer

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.Diagnostics;

public class Timer : MonoBehaviour
{
    public Text timerText;
    Stopwatch run_timer = new Stopwatch();

    private bool isRunning = false;

    GameObject[] initialObjects;
    Vector3[] originalPosition;
    Vector3[] originalRotation;

    // Start is called before the first frame update
    public void SetUpTimer(int num_objects)
    {
        //Fetch the Collider from the GameObject this script is attached to
        initialObjects = GameObject.FindGameObjectsWithTag("Interactable");
        originalPosition = new Vector3[num_objects];
        originalRotation = new Vector3[num_objects];
        for (int i = 0; i < num_objects; i++)
        {
            originalPosition[i] = initialObjects[i].transform.position;
            originalRotation[i] = initialObjects[i].transform.rotation.eulerAngles;
        }
    }
    void ResetInteractables()
    {
        for (int i = 0; i < 1; i++)
        {
            initialObjects[i].transform.position = originalPosition[i];
            initialObjects[i].transform.eulerAngles = originalRotation[i];
            initialObjects[i].GetComponent<Rigidbody>().isKinematic = true;
        }
    }
}

```



```

    }
}

public void TimerStart()
{
    if (!isRunning)
    {
        isRunning = true;
        run_timer.Start();
    }
}

public void TimerStop()
{
    if (isRunning)
    {
        isRunning = false;
        run_timer.Stop();
    }
}

void Update()
{
    if (isRunning)
    {
        timerText.text = run_timer.Elapsed.ToString(@"mm\:ss\.ff");
    }
}
}

```

Algorithm C-9. SlidersManager

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class SlidersManager : MonoBehaviour
{
    //Make sure to assign this in the Inspector window
    public GameObject collidercount;
    GameObject container;
    CollidersCounter colliders_counter;
    public float Weight;
}

```

```

public float Radiation;
public float PackingEfficiency;
public float centerOfGravity;
Slider weightslider;
Slider radioactivityslider;
Slider volumeslider;
Slider gravityslider;
Text weightText;
Text radiationText;
Text efficiencyText;
Text cogErrorText;
public Text nameOfVolunteer;
private int frames = 0;
private int countOfpackedObjects;
private float COGerror;
private float largesterror;
private ConfigReporter reporter;
bool packing = false;
string mode = "linear";

void Awake()
{
    Weight = 0;
    Radiation = 0;
    PackingEfficiency = 0;
    countOfpackedObjects = 0;
    nameOfVolunteer = GetComponent<Text>();

    colliders_counter = collidercount.GetComponent<CollidersCounter>();
    container = collidercount;

    reporter = GameObject.Find("Controller").GetComponent<ConfigReporter>();

    if (SceneManager.GetActiveScene().name == "Trial-and-Error" ||
        SceneManager.GetActiveScene().name == "Autonomous Packing")
    {
        mode = "trial";
    } else if (SceneManager.GetActiveScene().name == "Linear Packing")
    {
        mode = "linear";
    }
}

if (mode == "trial")
{
    weightslider = GameObject.Find("WeightSlider").GetComponent<Slider>();
    radioactivityslider = GameObject.Find("RadiationSlider").GetComponent<Slider>();
    volumeslider = GameObject.Find("EfficiencySlider").GetComponent<Slider>();
    gravityslider = GameObject.Find("COGError").GetComponent<Slider>();
}

```

```

    }
    else if (mode == "linear")
    {
        weightText = GameObject.Find("WeightPercentage").GetComponent<Text>();
        radiationText = GameObject.Find("RadiationPercentage").GetComponent<Text>();
        efficiencyText = GameObject.Find("EfficiencyPercentage").GetComponent<Text>();
        cogErrorText = GameObject.Find("COGErrorPercentage").GetComponent<Text>();
    }
}

void Update()
{
    colliders_counter = collidercount.GetComponent<CollidersCounter>();
    container = collidercount;
    centerOfGravity = 0;
    COGError = 0;
    largesterror = (container.transform.lossyScale.x + container.transform.lossyScale.y +
container.transform.lossyScale.z) / 2;
    Weight = colliders_counter.Totalweight / colliders_counter.weightLimitation;
    Radiation = colliders_counter.Totalradioactivity / colliders_counter.doseLimitation;
    PackingEfficiency = colliders_counter.Totalvolume / colliders_counter.colliderVolume;

    countOfpackedObjects = colliders_counter.count;

    if (colliders_counter.centerOfGravity.y != 0)
    {
        COGError = Mathf.Abs(colliders_counter.centerOfGravity.y - container.transform.position.y)
+ Mathf.Abs(colliders_counter.centerOfGravity.x - container.transform.position.x) +
Mathf.Abs(colliders_counter.centerOfGravity.z - container.transform.position.z);
        centerOfGravity = COGError / largesterror;
    }

    if (mode == "trial")
    {
        weightslider.value = Weight;
        radioactivityslider.value = Radiation;
        volumeslider.value = PackingEfficiency;
        gravityslider.value = centerOfGravity;
    }
    else if (mode == "linear")
    {
        weightText.GetComponent<ShowValueScript>().textUpdate(Weight);
        radiationText.GetComponent<ShowValueScript>().textUpdate(Radiation);
        efficiencyText.GetComponent<ShowValueScript>().textUpdate(PackingEfficiency);
        cogErrorText.GetComponent<ShowValueScript>().textUpdate(centerOfGravity);
    }
}

```

```

frames++;
if (frames % 50 == 0)
{
    //If the remainder of the current frame divided by 200 is 0 run the function.
    reporter.AppendToReport(GetReportLine());
}
}

public void SetTextOpacity()
{
    if (!packing)
    {
        weightText.GetComponent<CanvasGroup>().alpha = 0;
        radiationText.GetComponent<CanvasGroup>().alpha = 0;
        efficiencyText.GetComponent<CanvasGroup>().alpha = 0;
        cogErrorText.GetComponent<CanvasGroup>().alpha = 0;
        packing = true;
    }
    else if (packing)
    {
        weightText.GetComponent<CanvasGroup>().alpha = 1;
        radiationText.GetComponent<CanvasGroup>().alpha = 1;
        efficiencyText.GetComponent<CanvasGroup>().alpha = 1;
        cogErrorText.GetComponent<CanvasGroup>().alpha = 1;
        packing = false;
    }
}

string[] GetReportLine()
{
    string[] returnable = new string[6];
    returnable[0] = PackingEfficiency.ToString();
    returnable[1] = centerOfGravity.ToString();
    returnable[2] = GameObject.Find("Timer").GetComponent<Timer>().timerText.text;
    returnable[3] = Radiation.ToString();
    returnable[4] = Weight.ToString();
    returnable[5] = countOfpackedObjects.ToString();
    return returnable;
}
}

```

Algorithm C-10. ObjectManagerGA

```

using Plawius.NonConvexCollider;
using System.Collections.Generic;
using BzKovSoft.ObjectSlicer.Samples;
using Plawius.NonConvexCollider.Editor;
using UnityEngine;

```

```

using UnityEngine.UI;
using NewtonVR;
using UnityEditor.Sprites;

public class ObjectManagerGA : MonoBehaviour
{
    [HideInInspector] public List<string> selections = new List<string>();
    private bool ga_ran = false;
    private NVRPlayer player = null;
    public GameObject lid;

    private CollidersCounter counter;
    private SelectionManager selection_manager;
    private BLFPacking packer;
    private BLFPackingInwards inwardsPacker;
    private GAController ga_controller;
    private ConfigReporter reporter;

    public Timer t;

    public List<GameObject> Wastes;

    private void Awake()
    {
        selection_manager = gameObject.GetComponent<SelectionManager>();
        packer = gameObject.GetComponent<BLFPacking>();
        inwardsPacker = gameObject.GetComponent<BLFPackingInwards>();
        ga_controller = gameObject.GetComponent<GAController>();
        reporter = gameObject.GetComponent<ConfigReporter>();

        player = NVRPlayer.Instance;

        if ( player == null )
        {
            Debug.LogError( "Teleport: No Player instance found in map." );
            Destroy( this.gameObject );
            return;
        }

        lid = GameObject.Find("wall6");
        lid.SetActive(false);
    }

    void Update()
    {
        if (player.RightHand.Inputs[NVRButtons.Touchpad].IsPressed)
        {

```

```

var interacting = player.RightHand.CurrentlyInteracting;
if (interacting != null && !selections.Contains(interacting.name))
{
    selection_manager.SelectObject(interacting.gameObject);
}
selections = selection_manager.GetSelections();
}
else if (player.LeftHand.Inputs[NVRButtons.Touchpad].IsPressed)
{
    var interacting = player.LeftHand.CurrentlyInteracting;
    if (interacting != null && selections.Contains(interacting.name))
    {
        selection_manager.DeselectObject(interacting.gameObject);
    }
    selections = selection_manager.GetSelections();
}
}

if (!ga_ran && ga_controller.Finished())
{
    gameObject.GetComponent<ConfigReporter>().SetUp();
    Wastes = new List<GameObject>(GameObject.FindGameObjectsWithTag("Interactable"));
    for(int i = 0; i < Wastes.Count; i++)
    {
        if (selections.Contains(Wastes[i].name))
        {
            //selection_manager.DeselectObject(Wastes[i]);
            selections.Remove(Wastes[i].name);
        }
    }
    lid.SetActive(false);
    ga_ran = true;
}
}

public List<string> GetSelections()
{
    return selections;
}

public void RunGA()
{
    Debug.LogFormat("SELECTED FOR GA: {0}", selections.Count);
    lid.SetActive(true);
    inwardsPacker.SetUp(selections);
    ga_controller.SetUp(selections.Count);
    ga_controller.StartGA();
}

```

```

void AddNonConvexColliders(Mesh m, GameObject selected_object)
{
    Mesh[] meshes;
    meshes = API.GenerateConvexMeshes(m, Parameters.Default());

    var collider_asset = NonConvexColliderAsset.CreateAsset(meshes);
    var non_convex = selected_object.AddComponent<NonConvexColliderComponent>();
    non_convex.SetPhysicsCollider(collider_asset);
}
}

```

Algorithm C-11. GAController

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Diagnostics;
using GeneticSharp.Domain.Crossovers;
using GeneticSharp.Domain.Mutations;
using GeneticSharp.Domain.Populations;
using GeneticSharp.Domain.Selections;
using GeneticSharp.Domain.Chromosomes;
using GeneticSharp.Domain.Randomizations;
using GeneticSharp.Domain.Reinsertions;
using UnityEngine;
using Debug = UnityEngine.Debug;

public class GAController : MonoBehaviour
{
    private Thread m_gaThread;
    private GeneticAlgorithm m_ga;

    private int num_obj;

    private int m_currentGenerationNumber;

    //Parameters of the Genetic Algorithm (can be changed)
    public int m_populationSize = 50;
    public int m_generationNumber = 30;
    public float p_mutation = 0.2f;
    public float p_crossover = 0.75f;

    private Population;
    private EliteSelection Selection;
    private PackingOrderBasedCrossover Crossover;
    private PackingFlipBitMutation Mutation;
    private ElitistReinsertion Reinsertion;
}

```

```

private bool has_started = false;
private bool finished = false;

Stopwatch watch = new Stopwatch();
Stopwatch GAwatch = new Stopwatch();

private PackingChromosome best;

private bool initialSetupMethods = false;

#region Methods

public void SetUp(int num_objects)
{
    num_obj =
num_objects;//GameObject.FindGameObjectsWithTag("Interactable").Length;
    //PackingChromosome adam = new PackingChromosome(num_obj, 0,
num_obj,gameObject.GetComponent<BLFPacking>());
    PackingChromosome adam = new PackingChromosome(num_obj, 0, num_obj,
gameObject.GetComponent<BLFPackingInwards>());
    Population = new Population(m_populationSize, m_populationSize, adam);
    Selection = new EliteSelection();
    Crossover = new PackingOrderBasedCrossover();
    Mutation = new PackingFlipBitMutation();
    Reinsertion = new ElitistReinsertion();
}

private void Update()
{
    if (has_started && !finished) //Where all the Genetic Algorithm process happens
    {

        if (Population.GenerationsNumber == 0)
        {
            GAwatch.Start();
            Population.CreateInitialGeneration();
        }

        if (Population.GenerationsNumber <= m_generationNumber)
        {
            Debug.LogFormat("GENERATION #{0}",
Population.GenerationsNumber);
            watch.Start();
            foreach (PackingChromosome c in
Population.CurrentGeneration.Chromosomes)
            {
                c.Evaluate(false); //Is called for each chromosome
            }
        }
    }
}

```



```

        /*Debug.Log("-----CHROMOSOME-----");
        c.PrintChromosome();
        Debug.Log("CURRENT BEST FITNESS: " + c.Fitness);
        Debug.Log("-----");*/
    }

    Population = EvolveOneGeneration(Population);
    best = (PackingChromosome)Population.BestChromosome;
    watch.Stop();

    Debug.Log("-----CURRENT BEST CHROMOSOME-----");
    best.PrintChromosome();
    Debug.Log("CURRENT BEST FITNESS: " + best.Fitness);
    Debug.Log("-----");
    Debug.Log("GENERATION EXECUTION TIME " +
watch.Elapsed.ToString(@"mm:ss.ff"));
    watch.Reset();
} else if (Population.GenerationsNumber > m_generationNumber &&
!finished)
{
    finished = true;
    GAwatch.Stop();

    Debug.Log("-----BEST CHROMOSOME-----");
    best.PrintChromosome();
    Debug.Log("BEST FITNESS: " + best.Fitness);
    Debug.Log("-----");
    Vector3 iResults = best.Evaluate(true); //Will run the BLF Packing
with the chromosome that has the best fitness

    Debug.Log("-----FINAL RESULTS-----");
    Debug.Log("# of generations: " + m_generationNumber + ", population size: " +
m_populationSize);

    string blfType;
    if (best.packer.outwards) blfType = "Outwards BLF";
    else blfType = "Inwards BLF";

    Debug.Log(blfType + "Results: 1.Pack. Eff.: " + iResults.z + ", 2. Num. of Obj packed: " +
best.packer.colliders_counter.count);
    Debug.Log("GA EXECUTION TIME " + GAwatch.Elapsed.ToString(@"hh:mm:ss.ff"));
}
}

public void StartGA() //Called to start the process of the genetic algorithm, started when the
button is pressed in the UI
{

```

```

        if (!has_started)
        {
            has_started = true;
        }
    }

    public bool Finished()
    {
        return finished;
    }

    private Population EvolveOneGeneration(Population p)
    {
        var parents = Selection.SelectChromosomes(p.MinSize, p.CurrentGeneration);

        var minSize = Population.MinSize;
        var offsprings = new List<IChromosome>(minSize);

        for (int i = 0; i < minSize; i += Crossover.ParentsNumber)
        {
            var selectedParents = parents.Skip(i).Take(Crossover.ParentsNumber).ToList();

            if (selectedParents.Count == Crossover.ParentsNumber &&
                RandomizationProvider.Current.GetDouble() <= p_crossover)
            {
                var children = Crossover.Cross(selectedParents);

                if (children != null)
                {
                    offsprings.AddRange(children);
                }
            }

            for (int i = 0; i < offsprings.Count; i++)
            {
                Mutation.Mutate(offsprings[i], p_mutation);
            }

            Reinsertion.SelectChromosomes(p, offsprings, parents);

            p.EndCurrentGeneration();
            p.CreateNewGeneration(offsprings);

            return p;
        }
    }
}
#endregion

```

```
}
```

Algorithm C-12. PackingChromosome

```
using UnityEngine;
using System;
using System.Linq;
using GeneticSharp.Infrastructure.Framework.Texts;
using GeneticSharp.Infrastructure.Framework.Commons;
using GeneticSharp.Domain.Chromosomes;
using GeneticSharp.Domain.Randomizations;

public struct ObjectStruct
{
    public int id;
    public GameObject obj;
    public Vector3 orig_pos;
    public Vector3 orig_rot;

    public ObjectStruct(int id_, GameObject obj_, Vector3 pos_, Vector3 rot_)
    {
        id = id_;
        obj = obj_;
        orig_pos = pos_;
        orig_rot = rot_;
    }
}

public sealed class PackingChromosome : IChromosome
{
    #region Fields
    private Gene[] m_genes;
    private int m_length;

    private readonly int m_minValue;
    private readonly int m_maxValue;
    public BLFPackingInwards packer;
    #endregion

    #region Constructors
    /// <summary>
    /// Initializes a new instance of the <see cref="ChromosomeBase"/> class.
    /// </summary>
    /// <param name="length">The length, in genes, of the chromosome.</param>
    public PackingChromosome(int length, int minValue, int maxValue, BLFPackingInwards packer_)
    {
        ValidateLength(length);

        m_length = length;
    }
}
```

```

    m_genes = new Gene[length];
    m_minValue = minValue;
    m_maxValue = maxValue;
    packer = packer_;
    CreateGenes();
}
#endregion

#region Properties

public double? Fitness { get; set; }

public int Length
{
    get
    {
        return m_length;
    }
}
#endregion

#region Methods
public Gene GenerateGene(int geneIndex)
{
    int[] geneValue = new int[4];

    // object ID
    geneValue[0] = geneIndex; //RandomizationProvider.Current.GetInt(m_minValue,
m_maxValue+1);

    // ry bits
    geneValue[1] = RandomizationProvider.Current.GetInt(0, 2);
    geneValue[2] = RandomizationProvider.Current.GetInt(0, 2);
    geneValue[3] = RandomizationProvider.Current.GetInt(0, 2);
    return new Gene(geneValue);
}

public Gene GenerateSpecificGene(int geneIndex, int b1, int b2, int b3)
{
    int[] geneValue = {geneIndex, b1, b2, b3};
    return new Gene(geneValue);
}

IChromosome IChromosome.CreateNew()
{
    return CreateNew();
}
public PackingChromosome CreateNew()

```

```

    {
        return new PackingChromosome(m_length, m_minValue, m_maxValue, packer);
    }

    public IChromosome Clone()
    {
        var clone = CreateNew();
        clone.ReplaceGenes(0, GetGenes());
        clone.Fitness = Fitness;

        return clone;
    }
    public void ReplaceGene(int index, Gene gene)
    {
        if (index < 0 || index >= m_length)
        {
            throw new ArgumentOutOfRangeException(nameof(index), "There is no Gene on index {0}
to be replaced.".With(index));
        }

        m_genes[index] = gene;
        Fitness = null;
    }

    public void ReplaceGenes(int startIndex, Gene[] genes)
    {
        ExceptionHelper.ThrowIfNull("genes", genes);

        if (genes.Length > 0)
        {
            if (startIndex < 0 || startIndex >= m_length)
            {
                throw new ArgumentOutOfRangeException(nameof(startIndex), "There is no Gene on
index {0} to be replaced.".With(startIndex));
            }

            var genesToBeReplacedLength = genes.Length;

            var availableSpaceLength = m_length - startIndex;

            if (genesToBeReplacedLength > availableSpaceLength)
            {
                throw new ArgumentException(
                    nameof(Gene),
                    "The number of genes to be replaced is greater than available space, there is {0} genes
between the index {1} and the end of chromosome, but there is {2} genes to be replaced."
                    .With(availableSpaceLength, startIndex, genesToBeReplacedLength));
            }
        }
    }

```

```

        Array.Copy(genes, 0, m_genes, startIndex, genes.Length);

        Fitness = null;
    }
}

public void Resize(int newLength)
{
    ValidateLength(newLength);

    Array.Resize(ref m_genes, newLength);
    m_length = newLength;
}

public Gene GetGene(int index)
{
    return m_genes[index];
}

public Gene[] GetGenes()
{
    return m_genes;
}

public int CompareTo(IChromosome other)
{
    if (other == null)
    {
        return -1;
    }

    var otherFitness = other.Fitness;

    if (Fitness == otherFitness)
    {
        return 0;
    }

    return Fitness > otherFitness ? 1 : -1;
}

public override bool Equals(object obj)
{
    var other = obj as IChromosome;

    if (other == null)
    {
        return false;
    }
}

```

```

    return CompareTo(other) == 0;
}

public override int GetHashCode()
{
    return Fitness.GetHashCode();
}

private void CreateGene(int index)
{
    ReplaceGene(index, GenerateGene(index));
}

private void CreateGenes()
{
    var obj_indexes = RandomizationProvider.Current.GetUniqueInts(Length, m_minValue,
m_maxValue);
    for (int i = 0; i < Length; i++)
    {
        ReplaceGene(i, GenerateGene(obj_indexes[i]));
    }
}

private static void ValidateLength(int length)
{
    if (length < 2)
    {
        throw new ArgumentException("The minimum length for a chromosome is 2 genes.",
nameof(length));
    }
}

public Vector3 Evaluate(bool print)
{
    packer.SetupPacking(this);
    Vector3 properties = packer.RunBLFAbstract();

    if (properties.x > 1f && properties.y > 1f)
    {
        if (print)
        {
            Debug.Log("WEIGHT AND RADIATION EXCEEDED");
        }
        properties.z -= 0.15f;
    } else if (properties.x > 1f || properties.y > 1f)
    {
        if (print && properties.x > 1f)

```

```

    {
        Debug.Log("WEIGHT EXCEEDED");
    } else if (print && properties.y > 1f)
    {
        Debug.Log("RADIATION EXCEEDED");
    }
    properties.z -= 0.1f;
}

if (print) Debug.Log(properties);
this.Fitness = properties.z;
return properties;
}

public void FlipSpecificGeneValues(int index, int[] flip_indexes)
{
    var value = GetGene (index).Value as int[];

    int[] new_bits = new int[3];
    for (int i = 0; i < 3; i++)
    {
        if (flip_indexes.Contains(i))
        {
            new_bits[i] = value[1+i] == 0 ? 1 : 0;
        }
        else
        {
            new_bits[i] = value[1+i];
        }
    }

    Gene new_gene = GenerateSpecificGene(value[0], new_bits[0], new_bits[1], new_bits[2]);
    var print_gene = new_gene.Value as int[];

    ReplaceGene (index, new_gene);
}

public void FlipGene (int index)
{
    var value = GetGene (index).Value as int[];
    int b1 = value[1] == 0 ? 1 : 0;
    int b2 = value[2] == 0 ? 1 : 0;
    int b3 = value[3] == 0 ? 1 : 0;

    Gene new_gene = GenerateSpecificGene(value[0], b1, b2, b3);

    ReplaceGene (index, new_gene);
}

```



```

public void PrintChromosome()
{
    string print_msg = "\n";
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < this.Length; j++)
        {
            int[] geneValue = this.GetGene(j).Value as int[];
            print_msg += geneValue[i];
            if (j < this.Length - 1)
            {
                print_msg += ",";
            }
        }
        print_msg += "\n";
    }

    Debug.Log(print_msg);
}

#endregion
}

```

Algorithm C-13. PackingOrderBasedCrossover

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using GeneticSharp.Domain.Chromosomes;
using GeneticSharp.Domain.Randomizations;
using UnityEngine;

namespace GeneticSharp.Domain.Crossovers
{
    /// <summary>
    /// Order-based crossover (OX2).
    /// <remarks>
    /// OX2 was suggested in connection with schedule problems, is a modification of the OX1
    operator.
    /// The OX2 operator selects at random several positions in a parent string, and the order of the
    elements in the
    /// selected positions of this parent are imposed on the other parent. For example, consider the
    parents
    /// (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 1), and suppose that in the second parent in the second, third
    /// and sixth positions are selected. The elements in these positions are 4, 6 and 5 respectively.

```

```

    /// In the first parent, these elements are present at the fourth, fifth and sixth positions.
    /// Now the offspring are equal to parent 1 except in the fourth, fifth and sixth positions: (1 2 3 * *
* 7 8).
    /// We add the missing elements to the offspring in the same order in which they appear in the
second parent.
    /// This results in (1 2 3 4 6 5 7 8). Exchanging the role of the first parent and the second parent
gives,
    /// using the same selected positions, (2 4 3 8 7 5 6 1).
    /// </remarks>
    /// </summary>
    [DisplayName("Order-based (OX2)")]
    public class PackingOrderBasedCrossover : CrossoverBase
    {
        #region Constructors

        /// <summary>
        /// Initializes a new instance of the <see
cref="GeneticSharp.Domain.Crossovers.OrderBasedCrossover"/> class.
        /// </summary>
        public PackingOrderBasedCrossover()
            : base(2, 2)
        {
            IsOrdered = true;
        }

        #endregion

        #region Methods

        /// <summary>
        /// Performs the cross with specified parents generating the children.
        /// </summary>
        /// <param name="parents">The parents chromosomes.</param>
        /// <returns>The offspring (children) of the parents.</returns>
        protected override IList<IChromosome> PerformCross(IList<IChromosome> parents)
        {
            ValidateParents(parents);

            var parentOne = parents[0] as PackingChromosome;
            var parentTwo = parents[1] as PackingChromosome;

            var rnd = RandomizationProvider.Current;
            var swapIndexesLength = rnd.GetInt(1, parentOne.Length - 1);
            var swapIndexes = rnd.GetUniqueInts(swapIndexesLength, 0, parentOne.Length);

            // TODO: Choose random objects to crossover rotations than crossing over between already
crossed objects

```

```

var firstChild = CreateChild(parentOne, parentTwo, swapIndexes);
var secondChild = CreateChild(parentTwo, parentOne, swapIndexes);

return new List<IChromosome>() { firstChild, secondChild };
}

/// <summary>
/// Validates the parents.
/// </summary>
/// <param name="parents">The parents.</param>
protected virtual void ValidateParents(ICollection<IChromosome> parents)
{
    if (parents.AnyHasRepeatedGene())
    {
        throw new CrossoverException(this, "The Order-based Crossover (OX2) can be only used
with ordered chromosomes. The specified chromosome has repeated genes.");
    }
}

/// <summary>
/// Creates the child.
/// </summary>
/// <param name="firstParent">First parent.</param>
/// <param name="secondParent">Second parent.</param>
/// <param name="swapIndexes">The swap indexes.</param>
/// <returns>
/// The child.
/// </returns>
protected virtual IChromosome CreateChild(IChromosome firstParent, IChromosome
secondParent, int[] swapIndexes)
{
    // ...suppose that in the second parent in the second, third
    // and sixth positions are selected. The elements in these positions are 4, 6 and 5 respectively...
    var secondParentSwapGenes = secondParent.GetGenes()
        .Select((g, i) => new { Gene = g, Index = i })
        .Where((g) => swapIndexes.Contains(g.Index))
        .ToArray();

    var firstParentGenes = firstParent.GetGenes();

    // ...in the first parent, these elements are present at the fourth, fifth and sixth positions...
    var firstParentSwapGenes = firstParentGenes
        .Select((g, i) => new { Gene = g, Index = i })
        .Where((g) => secondParentSwapGenes.Any(s => (s.Gene.Value as int[])[0] ==
(g.Gene.Value as int[])[0]))
        .ToArray();

    var child = (PackingChromosome)firstParent.CreateNew();

```

```

var secondParentSwapGensIndex = 0;

for (int i = 0; i < firstParent.Length; i++)
{
    // Now the offspring are equal to parent 1 except in the fourth, fifth and sixth positions.
    // We add the missing elements to the offspring in the same order in which they appear in
the second parent.
    if (firstParentSwapGens.Any(f => f.Index == i))
    {
        var gene1 = firstParentGenes[i].Value as int[];
        var gene2 = secondParentSwapGens[secondParentSwapGensIndex++].Gene.Value as
int[];
        var new_gene = CrossAngleBits(gene1, gene2);
        child.ReplaceGene(i, new
Gene(new_gene)); //secondParentSwapGens[secondParentSwapGensIndex++].Gene);
    }
    else
    {
        child.ReplaceGene(i, firstParentGenes[i]);
    }
}

return child;
}

int[] CrossAngleBits(int[] g1, int[] g2)
{
    int[] c = new int[4];
    c[0] = g2[0];
    c[1] = g1[1] ^ g2[1];
    c[2] = g1[2] ^ g2[2];
    c[3] = g1[3] ^ g2[3];
    return c;
}

#endregion
}
}

```

Algorithm C-14. PackingFlipBitMutation

```

using System;
using System.ComponentModel;
using GeneticSharp.Domain.Chromosomes;
using GeneticSharp.Domain.Randomizations;
using UnityEngine;

namespace GeneticSharp.Domain.Mutations

```

```

{
  /// <summary>
  /// Takes the chosen genome and inverts the bits (i.e., if the genome bit is 1, it is changed to 0 and
  vice versa).
  /// </summary>
  /// <remarks>
  /// When using this mutation, the genetic algorithm should use IBinaryChromosome.
  /// </remarks>
  [DisplayName("Flip Bit")]
  public class PackingFlipBitMutation : MutationBase
  {
    #region Fields
    private readonly IRandomization m_rnd;
    #endregion

    #region Constructors
    /// <summary>
    /// Initializes a new instance of the <see
    cref="GeneticSharp.Domain.Mutations.FlipBitMutation"/> class.
    /// </summary>
    public PackingFlipBitMutation ()
    {
      m_rnd = RandomizationProvider.Current;
    }
    #endregion

    #region Methods
    /// <summary>
    /// Mutate the specified chromosome.
    /// </summary>
    /// <param name="chromosome">The chromosome.</param>
    /// <param name="probability">The probability to mutate each chromosome.</param>
    protected override void PerformMutate (IChromosome chromosome, float probability)
    {
      var packingChromosome = chromosome as PackingChromosome;

      if (packingChromosome == null)
      {
        throw new MutationException (this, "Needs a packing chromosome that implements
        PackingChromosome.");
      }

      if (m_rnd.GetDouble() <= probability)
      {
        var index = m_rnd.GetInt(0, packingChromosome.Length);
        // Number of bits to flip
        var num_mutate = m_rnd.GetInt(1, 4);
        // indexes to flip (based on the number of bits to flip)

```

```

        int[] mutate_indexes = m_rnd.GetUniqueInts(num_mutate, 0, 3);
        packingChromosome.FlipSpecificGeneValues(index, mutate_indexes);
    }
}
#endregion
}
}

```

Algorithm C-15. SelectionManager

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SelectionManager : MonoBehaviour
{
    private GameObject selection = null;
    [SerializeField] private Material highlightMaterial;
    [SerializeField] private Material defaultMaterial;
    [SerializeField] private string selectableTag = "Selectable";

    public void SelectObject()
    {
        var ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit))
        {
            GameObject selected = hit.collider.gameObject;
            if (selected.CompareTag(selectableTag))
            {
                selection = selected;
                var selectionRenderer = selection.GetComponentInChildren<Renderer>();
                if (selectionRenderer != null)
                {
                    selectionRenderer.material = highlightMaterial;
                }
            }
        }
    }

    public void DeselectObject()
    {
        var selectionRenderer = selection.GetComponentInChildren<Renderer>();
        selectionRenderer.material = defaultMaterial;
        selection = null;
    }

    public GameObject GetSelection()

```

```

    {
        return selection;
    }
}

```

Algorithm C-16. BLFPackingInwards

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BLFPackingInwards : MonoBehaviour
{
    #region Variables
    public bool fullWorkflow;

    public bool outwards = false;

    public float cubeSize;
    public GameObject box;
    private int max_iter = 10;
    public CollidersCounter colliders_counter;

    [HideInInspector] public bool checkBetterPositions = false;
    [HideInInspector] public bool wiggleRoomBool = false;
    [HideInInspector] public bool rotateMeshBounds = false;

    // Object list used to keep track of object information including (id, original transformation)
    public List<ObjectStruct> object_list = new List<ObjectStruct>();

    // Object list used in BLF algorithm
    public List<GameObject> packing_objs = new List<GameObject>();

    private float[] Angles = new float[8] { 0f, 45f, 90f, 135f, 180f, 225f, 270f, 315f };

    public GameObject wallSix;

    public GameObject objOfInterest;

    #endregion

    #region Setup Methods

    public void SetUp(List<string> selected_objects)
    {
        GameObject[] objects = GameObject.FindGameObjectsWithTag("Interactable");
        packing_objs = new List<GameObject>(selected_objects.Count);
    }

```

```

    for (int i = 0; i < objects.Length; i++)
    {
        if (selected_objects.Contains(objects[i].name))
        {
            ObjectStruct obj = new ObjectStruct(i, objects[i], objects[i].transform.position,
objects[i].transform.eulerAngles);
            object_list.Add(obj);
        }
    }

    if (!fullWorkflow) colliders_counter =
GameObject.Find("Counter").GetComponent<CollidersCounter>();
}

public void SetupPacking(PackingChromosome chromosome)
{
    packing_objs.Clear();

    int ry;
    //int rx;
    //int rz;
    for (int i = 0; i < chromosome.Length; i++)
    {
        int[] gene = chromosome.GetGene(i).Value as int[];
        //Debug.Log(gene[0]);
        GameObject obj = object_list[gene[0]].obj; //index out of range
        packing_objs.Add(obj);
        ry = gene[1] + gene[2] * 2 + gene[3] * 2 * 2;
        //rx = gene[4] + gene[5] * 2 + gene[6] * 2 * 2;
        //rz = gene[7] + gene[8] * 2 + gene[9] * 2 * 2;
        obj.transform.eulerAngles = new Vector3(object_list[gene[0]].orig_rot.x,
object_list[gene[0]].orig_rot.y + Angles[ry], object_list[gene[0]].orig_rot.z);
        obj.transform.position = object_list[gene[0]].orig_pos;
    }
}

#endregion

void Update()
{
    if(objOfInterest == null) return;
}

public Vector3 RunBLFAbstract()
{
    return RunBLF(outwards);
}

```



```

//Packages the objects and returns the weight/radiation/packingefficiency values.
// If outwards = true, runs the outwards algorithm (starting from the bottom left front corner and
moving outwards)
// If outwards = false, starts the object from the opposite diagonal corner and moves it inwards until
it collides
public Vector3 RunBLF(bool outwards)
{
    Debug.Log("BOX NAME: " + box.name);
    if (outwards) PackOutwards(box, packing_objs, 0.01f, 0.01f, 0.01f);
    else PackInwards(box, packing_objs, 0.01f, 0.01f, 0.01f);

    colliders_counter.CalculateTotalProperties();
    float Weight = colliders_counter.Totalweight / colliders_counter.weightLimitation;
    float Radiation = colliders_counter.Totalradioactivity / colliders_counter.doseLimitation;
    float PackingEfficiency = colliders_counter.Totalvolume / colliders_counter.colliderVolume;

    return new Vector3(Weight, Radiation, PackingEfficiency);
}

void PackOutwards(GameObject box, List<GameObject> object_list, float x_offset, float y_offset,
float z_offset)
{
    Vector3 box_blfposition = box.GetComponent<BoxCollider>().transform.position; //Stores the
position of the box that the objects are placed in
    Vector3 box_size = box.GetComponent<BoxCollider>().size;
    box_size = Vector3.Scale(box_size, box.transform.localScale); //Resizes the box's collider
depending on its local scale

    //Box's (Counter game object) outwards and inwards corner's position in all three dimensions.
    //The outwards (far) corner is where the iteration begins. The inwards corner is where all the
objects are moving towards.
    Vector3 boxOutwardsCorner = box_blfposition + (box_size / 2);
    Vector3 boxInwardsCorner = box_blfposition - (box_size / 2);

    float x;
    float y;
    float z;
    float init_x;
    float init_y;
    float init_z;
    int num_max_reached = 0;

    foreach (GameObject curr_obj in object_list)
    {
        Vector3 pos_outside = curr_obj.transform.position;
        Vector3 prev_pos;
        int iter = 1;

```

```

Vector3 initialPosition = boxInwardsCorner;
Vector3 maximumPosition = boxOutwardsCorner;

while (iter <= max_iter)
{
    prev_pos = curr_obj.transform.position;
    if (iter == 1)
    {
        //Place object in the initial position, fix the position with CheckInitialCollision, and then
set the fixed position as the initial position
        curr_obj.transform.position = initialPosition;
        CheckInitialCollisionOutwards(curr_obj); //-> different for outwards
        initialPosition = curr_obj.transform.position;

        //Place object in the maximum position, fix the position with
CheckInitialCollisionInwards, and then set the fixed position as the initial position
        curr_obj.transform.position = maximumPosition;
        CheckInitialCollisionInwards(curr_obj); //-> different for outwards
        maximumPosition = curr_obj.transform.position;

        maximumPosition = new Vector3(maximumPosition.x, maximumPosition.y + 0.01f,
maximumPosition.z); //This modification allows packing 27 control cubes instead of 18

        x = initialPosition.x;
        z = initialPosition.z;
    }
    else
    {
        x = curr_obj.transform.position.x;
        z = curr_obj.transform.position.z;
    }

    // Bottom
    y = initialPosition.y;
    curr_obj.transform.position = new Vector3(x, y, z);
    while (CheckCollision(curr_obj) == 1)
    {
        y += y_offset;
        if (y >= maximumPosition.y)
        {
            // Restore the best y position
            y -= y_offset;
            num_max_reached++;
            break;
        }
    }
    curr_obj.transform.position = new Vector3(x, y, z);
}

```

```

}

// Left
x = initialPosition.x;
curr_obj.transform.position = new Vector3(x, y, z);
while (CheckCollision(curr_obj) == 1)
{
    x += x_offset;
    if (x > maximumPosition.x)
    {
        // Restore the best x position
        x -= x_offset;
        num_max_reached++;
        break;
    }
    curr_obj.transform.position = new Vector3(x, y, z);
}

// Front
z = initialPosition.z;
curr_obj.transform.position = new Vector3(x, y, z);
while (CheckCollision(curr_obj) == 1)
{
    z += z_offset;
    if (z > maximumPosition.z)
    {
        // Restore the best z position
        z -= z_offset;
        num_max_reached++;
        break;
    }
    curr_obj.transform.position = new Vector3(x, y, z);
}

// If object couldn't be packed within the limits, move it outside the box and continue to the
next object
if (num_max_reached == 3)
{
    curr_obj.transform.position = pos_outside;
    num_max_reached = 0;
    break;
}

// If there isn't an improvement compared to the previous position, continue to the next
object
if (Vector3.Distance(prev_pos, curr_obj.transform.position) < 0.01) break;
num_max_reached = 0;

```

```

        iter++;
    }
}
}

void PackInwards(GameObject box, List<GameObject> object_list, float x_offset, float y_offset,
float z_offset)
{
    wallSix.SetActive(true);

    Vector3 box_blfposition = box.GetComponent<BoxCollider>().transform.position; //Stores the
position of the box that the objects are placed in
    Vector3 box_size = box.GetComponent<BoxCollider>().size;
    box_size = Vector3.Scale(box_size, box.transform.localScale); //Resizes the box's collider
depending on its local scale

    //Box's (Counter game object) outwards and inwards corner's position in all three dimensions.
    //The outwards (far) corner is where the iteration begins. The inwards corner is where all the
objects are moving towards.
    Vector3 boxOutwardsCorner = box_blfposition + (box_size / 2.00f);
    Vector3 boxInwardsCorner = box_blfposition - (box_size / 2.00f);

    foreach (GameObject curr_obj in object_list)
    {
        //Set transform to manipulate
        Transform manipulatedTransform = curr_obj.transform;
        bool pivotChanged = false;

        Vector3 pos_outside = manipulatedTransform.position; //outside position recorded, if box
can't be fit it will be placed here in the end
        Vector3 prev_pos; //stores the 1 previous position of the object
        int iter = 1;
        Vector3 initialPosition = boxOutwardsCorner;

        //Fix the initial position

        //Place them initially using the renderer bounds
        Vector3 obj_bounds = curr_obj.GetComponent<Renderer>().bounds.size;

        manipulatedTransform.position = boxOutwardsCorner
            - (obj_bounds / 2.00f); //Half of obj bounds because initially they are in
the centre
        Debug.Log("For " + curr_obj.name + " FIX: Before pushAway: " +
manipulatedTransform.position.x + ", "
            + manipulatedTransform.position.y + ", "
            + manipulatedTransform.position.z);
        PushObjectAwayFromWalls(curr_obj);
    }
}

```

```

    Debug.Log("For " + curr_obj.name + " FIX: After pushAway: " +
manipulatedTransform.position.x + ", "
                                + manipulatedTransform.position.y + ", "
                                + manipulatedTransform.position.z);
    initialPosition = manipulatedTransform.position;

//Place object, check further positions, if no further position and it is in collision move it out

while (iter <= max_iter)
{
    prev_pos = manipulatedTransform.position;
    Vector3 iteratedPosition = manipulatedTransform.position; //This Vector3 will store the
iterated positions throughout the algorithm

    int count = 0;
    // BOTTOM (y-axis)
    while (CheckCollision(curr_obj) == 0) //While the object is not colliding with anything
    {
        iteratedPosition.y -= y_offset;
        manipulatedTransform.position = iteratedPosition;
        //Debug.Log("Iterated position (y): " + manipulatedTransform.position);
        count++;

        if (count > 100)
        {
            Debug.Log("For " + curr_obj.name + " check collision reached 100 repetitions,
break.");
            break;
        }
    }

    iteratedPosition.y += y_offset; //The object had just collided, move it back 1 step (this is the
best position)
    manipulatedTransform.position = iteratedPosition;
    if (checkBetterPositions)
    {
        CheckFurtherPositions(curr_obj, "y", pivotChanged); //Check future possible positions in
the y-direction that might be blocked
        iteratedPosition = manipulatedTransform.position;
    }

    count = 0;
    // LEFT (x-axis)
    while (CheckCollision(curr_obj) == 0)
    {
        iteratedPosition.x -= x_offset;
        manipulatedTransform.position = iteratedPosition;
        count++;

```

```

        if (count > 100)
        {
            Debug.Log("For " + curr_obj.name + " check collision reached 100 repetitions,
break.");
            break;
        }
    }

    iteratedPosition.x += x_offset;
    manipulatedTransform.position = iteratedPosition;
    if (checkBetterPositions)
    {
        CheckFurtherPositions(curr_obj, "x", pivotChanged); //Check future possible positions in
the x-direction that might be blocked
        iteratedPosition = manipulatedTransform.position;
    }

    count = 0;
    // FRONT (z-axis)
    while (CheckCollision(curr_obj) == 0)
    {
        iteratedPosition.z -= z_offset;
        manipulatedTransform.position = iteratedPosition;
        count++;

        if (count > 100)
        {
            Debug.Log("For " + curr_obj.name + " check collision reached 100 repetitions,
break.");
            break;
        }
    }

    iteratedPosition.z += z_offset;
    manipulatedTransform.position = iteratedPosition;
    if (checkBetterPositions)
    {
        CheckFurtherPositions(curr_obj, "z", pivotChanged);
        //Check future possible positions in the z-direction that might be blocked
        iteratedPosition = manipulatedTransform.position;
    }

    if (Vector3.Distance(prev_pos, manipulatedTransform.position) < 0.01)
    {
        Debug.Log("For " + curr_obj.name + " no improvement found, break");
        break; //If there is no improvement, no need to continue running the algorithm
    }
}

```

```

        if (iter == max_iter) //After all the iterations, if the objects final position is WORSE than its
first one, move the object out
        {
            //Least points = better score (more towards the inner corner)
            float initPoints = initialPosition.x + initialPosition.y + initialPosition.z;
            float finalPoints = manipulatedTransform.position.x + manipulatedTransform.position.y
+ manipulatedTransform.position.z;
            if (initPoints <= finalPoints)
            {
                Debug.Log("For " + curr_obj.name + " final position was worse than first one, so
moved it out.");
                Debug.Log("For " + curr_obj.name + "init pos: " + initialPosition);
                Debug.Log("For " + curr_obj.name + "final pos: " + manipulatedTransform.position);
                manipulatedTransform.position = pos_outside;
            }
        }

        iter++;
    }

    curr_obj.GetComponent<Rigidbody>().isKinematic = false;
}

wallSix.SetActive(false);
}

void PushObjectAwayFromWalls(GameObject curr_obj)
{
    Vector3 iteratedPosition = curr_obj.transform.position;

    int count = 0;
    while(CheckWallCollision(curr_obj, new List<string> { "wall2" }) == 1)
    {
        iteratedPosition.z -= 0.01f;
        curr_obj.transform.position = iteratedPosition;

        count++;
        if (count > 100)
        {
            Debug.Log("For " + curr_obj.name + " reached max count here.");
            break;
        }
    }

    count = 0;
    while(CheckWallCollision(curr_obj, new List<string> { "wall3" }) == 1)
    {

```

```

    iteratedPosition.x -= 0.01f;
    curr_obj.transform.position = iteratedPosition;

    count++;
    if (count > 100)
    {
        Debug.Log("For " + curr_obj.name + " reached max count here.");
        break;
    }
}

count = 0;
while(CheckWallCollision(curr_obj, new List<string> { "wall6" }) == 1)
{
    iteratedPosition.y -= 0.01f;
    curr_obj.transform.position = iteratedPosition;

    count++;
    if (count > 100)
    {
        Debug.Log("For " + curr_obj.name + " reached max count here.");
        break;
    }
}
}
void CheckFurtherPositions(GameObject curr_obj, string direction, bool pivotChanged)
{
    Transform manipulatedTransform = curr_obj.transform;
    if (pivotChanged && curr_obj.transform.parent.name.Contains("(Pivot)"))
manipulatedTransform = curr_obj.transform.parent;

    Vector3 stopPoint = new Vector3();
    Vector3 bestPosition = curr_obj.transform.position;
    List<Vector3> possiblePositions = new List<Vector3>();
    Vector3 castDirection = new Vector3(0, 0, 0);
    float rayLength = 1f;

    //Set the direction, if the input is anything else break
    if (direction == "x")
        castDirection = new Vector3(-1, 0, 0);
    else if (direction == "y")
        castDirection = new Vector3(0, -1, 0);
    else if (direction == "z")
        castDirection = new Vector3(0, 0, -1);
    else
        return;

    //Cast a ray between where the curr_obj is and where the stopping wall will be

```



```

RaycastHit hit;
if (Physics.Raycast(manipulatedTransform.position, castDirection, out hit, rayLength,
LayerMask.GetMask("Ignore Raycast")))
{
    if (hit.collider != null && hit.collider.gameObject != curr_obj)
    {
        stopPoint = hit.point;
    }
}

//Generate points inbetween curr_obj and the wall at 0.01f distanced intervals
if (direction == "x")
{
    int numOfIterations = Mathf.RoundToInt((manipulatedTransform.position.x - hit.point.x) /
0.01f);
    for (int i = 0; i < numOfIterations; i++)
    {
        float xPos = manipulatedTransform.position.x - (0.01f * i);
        possiblePositions.Add(new Vector3(xPos, manipulatedTransform.position.y,
manipulatedTransform.position.z));

        if (wiggleRoomBool)
        {
            possiblePositions.Add(new Vector3(xPos, manipulatedTransform.position.y + 0.01f,
manipulatedTransform.position.z));
            possiblePositions.Add(new Vector3(xPos, manipulatedTransform.position.y - 0.01f,
manipulatedTransform.position.z));
            possiblePositions.Add(new Vector3(xPos, manipulatedTransform.position.y,
manipulatedTransform.position.z + 0.01f));
            possiblePositions.Add(new Vector3(xPos, manipulatedTransform.position.y,
manipulatedTransform.position.z - 0.01f));
        }
    }
}
else if (direction == "y")
{
    int numOfIterations = Mathf.RoundToInt((manipulatedTransform.position.y - hit.point.y) /
0.01f);
    for (int i = 0; i < numOfIterations; i++)
    {
        float yPos = manipulatedTransform.position.y - (0.01f * i);
        possiblePositions.Add(new Vector3(manipulatedTransform.position.x, yPos,
manipulatedTransform.position.z));

        if (wiggleRoomBool)
        {
            possiblePositions.Add(new Vector3(manipulatedTransform.position.x + 0.01f, yPos,
manipulatedTransform.position.z));

```

```

        possiblePositions.Add(new Vector3(manipulatedTransform.position.x - 0.01f, yPos,
manipulatedTransform.position.z));
        possiblePositions.Add(new Vector3(manipulatedTransform.position.x, yPos,
manipulatedTransform.position.z + 0.01f));
        possiblePositions.Add(new Vector3(manipulatedTransform.position.x, yPos,
manipulatedTransform.position.z - 0.01f));
    }
}
}
else if (direction == "z")
{
    int numOfIterations = Mathf.RoundToInt((manipulatedTransform.position.z - hit.point.z) /
0.01f);
    for (int i = 0; i < numOfIterations; i++)
    {
        float zPos = manipulatedTransform.position.z - (0.01f * i);
        possiblePositions.Add(new Vector3(manipulatedTransform.position.x,
manipulatedTransform.position.y, zPos));

        if (wiggleRoomBool)
        {
            possiblePositions.Add(new Vector3(manipulatedTransform.position.x + 0.01f,
manipulatedTransform.position.y, zPos));
            possiblePositions.Add(new Vector3(manipulatedTransform.position.x - 0.01f,
manipulatedTransform.position.y, zPos));
            possiblePositions.Add(new Vector3(manipulatedTransform.position.x,
manipulatedTransform.position.y + 0.01f, zPos));
            possiblePositions.Add(new Vector3(manipulatedTransform.position.x,
manipulatedTransform.position.y - 0.01f, zPos));
        }
    }
}

for (int i = 0; i < possiblePositions.Count; i++)
{
    manipulatedTransform.position = possiblePositions[i]; //Set object at possible position
    float totalBestPosition = bestPosition.x + bestPosition.y + bestPosition.z;
    float totalPossiblePosition = possiblePositions[i].x + possiblePositions[i].y +
possiblePositions[i].z;

    //if it doesn't collide and it is a better position, it is the best new position
    //better position = sum of x,y,z components are smaller
    if (CheckCollision(curr_obj) == 0 && totalPossiblePosition < totalBestPosition) bestPosition
= possiblePositions[i];
}

    manipulatedTransform.position = bestPosition;
}

```

```

#region Collision Check Methods

//This method checks if the obj is in collision with anything else
public int CheckCollision(GameObject obj)
{
    Vector3 obj_bounds =
Vector3.Scale(obj.GetComponent<MeshFilter>().sharedMesh.bounds.size, obj.transform.localScale);

    float r = Mathf.Max(obj_bounds.x, obj_bounds.y, obj_bounds.z);
    Vector3 c = obj.transform.position;

    List<Collider> colliding_objs = new List<Collider>(Physics.OverlapSphere(c, r));
    colliding_objs.RemoveAll(s => s.gameObject.name.Equals(obj.name) ||
s.gameObject.name.Equals(box.name));
    Collider[] obj_cols = obj.GetComponents<Collider>();

    foreach (Collider hitcollider in colliding_objs)
    {
        List<Collider> curr = new List<Collider>(colliding_objs.FindAll(s =>
s.gameObject.name.Equals(hitcollider.gameObject.name)));
        foreach (Collider curr_colliding in curr)
        {
            Vector3 dir;
            float dis;
            // Check collision for all colliders of the object
            foreach (Collider col in obj_cols)
            {
                if (Physics.ComputePenetration(col,
                    obj.transform.position,
                    obj.transform.rotation,
                    curr_colliding,
                    curr_colliding.transform.position,
                    curr_colliding.transform.rotation,
                    out dir,
                    out dis))
                {
                    return 1;
                }
            }
        }
    }
    return 0;
}

//This method checks if obj is colliding with any of the walls in the list, returns 1 if true, 0
otherwise
int CheckWallCollision(GameObject obj, List<string> wallNames)

```

```

{
    Vector3 obj_bounds =
Vector3.Scale(obj.GetComponent<MeshFilter>().sharedMesh.bounds.size, obj.transform.localScale);

    float r = Mathf.Max(obj_bounds.x, obj_bounds.y, obj_bounds.z);
    Vector3 c = obj.transform.position;

    List<Collider> colliding_objs = new List<Collider>(Physics.OverlapSphere(c, r));
    colliding_objs.RemoveAll(s => s.gameObject.name.Equals(obj.name) ||
s.gameObject.name.Equals(box.name));
    Collider[] obj_cols = obj.GetComponents<Collider>();

    foreach (Collider hitcollider in colliding_objs)
    {
        List<Collider> curr = new List<Collider>(colliding_objs.FindAll(s =>
s.gameObject.name.Equals(hitcollider.gameObject.name)));
        foreach (Collider curr_colliding in curr)
        {
            Vector3 dir;
            float dis;
            // Check collision for all colliders of the object
            foreach (Collider col in obj_cols)
            {
                if (Physics.ComputePenetration(col,
                    obj.transform.position,
                    obj.transform.rotation,
                    curr_colliding,
                    curr_colliding.transform.position,
                    curr_colliding.transform.rotation,
                    out dir,
                    out dis))
                {
                    {
                        if(wallNames.Contains(hitcollider.name))
                        {
                            return 1;
                        }
                    }
                }
            }
        }
    }
    return 0;
}
}

```