UNIVERSITY OF LEEDS

Cost-Optimised Cutting and Packing for Nuclear Decommissioning

by

Aron Oliver Webster

Submitted in accordance with the requirements for the degree of

Doctor of Philosophy in Chemical and Process Engineering

The University of Leeds

School of Chemical and Process Engineering

September 2025

Statement of Contributions

I confirm that the work submitted is my own, except where work which has formed part of jointly authored publications has been included. My contribution and the other authors to this work has been explicitly indicated below. I confirm that appropriate credit has been given within the thesis where reference has been made to the work of others.

Research Presented in Chapter 4, Section 4.2.3

A. Webster, X. Jia, and S. Q. Xie, 'Hyper-Heuristics for Irregular Object Multi-Container Packing', in *2023 29th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, Nov. 2023, pp. 1–4. doi: 10.1109/M2VIP58386.2023.10413433.

The project was conceptualised by Aron Webster. All algorithm development, MATLAB coding, data collection, and analysis was performed by Aron Webster. Xiaodong Jia modified the DigiPac code to interface with the MATLAB implementation. Manuscript preparation was led by Aron Webster, with supervisory guidance and editorial feedback provided by Xiaodong Jia and Shane Xie.

Research Presented in Chapter 5, Sections 5.1 & 5.2

A. Webster, X. Jia, and S. Q. Xie, 'Cost-Optimal Cutting and Packing for Nuclear Decommissioning', in *2024 30th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, Oct. 2024, pp. 1–6. doi: 10.1109/M2VIP62491.2024.10746009.

The project was conceptualised by Aron Webster. Implementation of the 3D cutting and packing algorithm into the NuPlant software was carried out by Xiaodong Jia under supervision and with input from Aron Webster. All data collection, experimental analysis, and manuscript drafting was performed by Aron Webster. Supervisory guidance and feedback on the manuscript was provided by Xiaodong Jia and Shane Xie.

Research Presented in Chapter 4, Section 4.2.2 (MSc Thesis Supervision)

Part of the work presented in Chapter 4 (relating to the single-container hyper-heuristic packing approach) was based on results originally obtained as part of an MSc dissertation supervised by Aron Webster:

J. Williams, A. Webster and X. Jia, 'Packing Optimisation with Hyper-Heuristics.', *Master's Thesis, University of Leeds, 2023*.

The project was conceptualised, defined and led by Aron Webster, who also developed and implemented the packing optimisation algorithm in MATLAB. Xiaodong Jia modified the existing DigiPac code to allow interfacing with the MATLAB code. The MSc student carried out the data collection and experimental analysis under the close supervision of Aron Webster and Xiaodong Jia.

Research Presented in Chapter 4, Section 4.4 (MSc Thesis Supervision)

Part of the work presented in Chapter 4 (relating to the disassembly sequencing study) was based on results originally obtained as part of an MSc dissertation supervised by Aron Webster:

A. Knight, A. Webster, X. Jia, 'Optimised Disassembly Sequencing for 2D Cutting and Packing', *Master's Thesis, University of Leeds, 2025*.

The project was conceptualised, defined and led by Aron Webster, who also developed and implemented the disassembly sequencing algorithms in MATLAB. The MSc student carried out the data collection and experimental analysis under the close supervision of Aron Webster and Xiaodong Jia. The results presented in Chapter 4 were adapted from the student's Dissertation, with additional observations being added by Aron Webster.

Acknowledgements

Firstly, I would like to express my sincerest gratitude to my supervisors, Dr. Xiaodong Jia and Professor Shane Xie, for their constant support, invaluable guidance, and encouragement throughout the course of this research project. I am especially grateful to my primary supervisor, Dr. Jia, for his extensive help during the 3D implementation of the algorithm; his technical input and coding support were instrumental to the success of this project. Beyond his academic mentorship, Dr. Jia has also been a generous and supportive guide, and I feel fortunate to have had the opportunity to work under someone so dedicated. His mentorship has not only shaped the direction of this research, but has also helped me grow into a more confident and independent researcher.

Secondly, I would also like to thank the MEng students who contributed to this work. In particular, Jack Williams and Adam Knight, who both provided valuable assistance for the hyper-heuristic and disassembly sequencing works (respectively) presented in this thesis. Their efforts and collaboration were greatly appreciated.

Additionally, I am deeply thankful to the GREEN (Growing skills for Reliable Economic Energy from Nuclear) Centre for Doctoral Training (CDT) programme and the Engineering and Physical Sciences Research Council (EPSRC) for funding this project and providing me with the opportunity to pursue this research. The training and resources made available through the CDT have been essential in shaping both the direction and impact of this work. Beyond this, the programme has also helped me gain a broader understanding and deeper appreciation of the nuclear sector, and has supported my development in key professional skills such as public speaking, science communication, and cross-disciplinary collaboration.

And finally, I'd like to thank my friends and family for their ongoing support, encouragement, and patience throughout this PhD journey.

Abstract

The cutting and packing of large, solid structures presents a significant challenge in nuclear decommissioning as these two objectives exist in a trade-off. Segmenting a structure into smaller parts can improve packing efficiency and reduce the number of containers required, but at the cost of increased cutting effort. Conversely, minimising cuts reduces cutting cost but often leads to poor packing, raising overall disposal costs. Determining the optimal balance between cutting and packing is difficult, as the trade-off is highly context-dependent, influenced by the geometry of the structure itself as well as the choice of cutting tools and container types used.

This thesis addresses the lack of integrated approaches capable of resolving this trade-off by proposing a novel computational framework that jointly optimises both cutting and packing to minimise total cost. A new feedback-driven optimisation strategy is proposed, in which a genetic algorithm (GA) iteratively refines cutting plans based on *both* the cutting cost and packing cost, allowing for dynamic exploration of trade-offs between conflicting objectives.

A simplified 2D prototype is developed to evaluate the proposed methodology for jointly optimising cutting and packing. Additional 2D studies are also undertaken to investigate ways to enhance the underlying cutting and packing processes, including hyper-heuristics for improving container utilisation, comparing multi-container packing strategies, and feasible disassembly sequencing for cut structures.

Building on insights gained from the 2D studies, the methodology is extended to 3D and applied to three representative nuclear decommissioning scenarios. Results show that the integrated algorithm can consistently outperform separate cutting/packing approaches (with cost reductions of up to 17%) and achieve comparable performance to manually designed strategies.

The thesis concludes with a detailed set of future work recommendations to enhance algorithm performance and real-world applicability, including the incorporation of feature-based segmentation, post-processing for packing refinement, and integration with robotic disassembly systems.

Table of Contents

List of Tables

List of Figures

Abbreviations

AI	Artificial Intelligence
ALARA	As Low as Reasonably Achievable
BSP	Binary Space Partitioning
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CiADS	China initiative Accelerator Driven System
DEM	Discrete Element Method
FFT	Fast Fourier Transform
FFRS	First Feasible Random Search
GA	Genetic Algorithm
GS	Greedy Search
GLS	Guided Local Search
HDS	Height Decreasing Search
IAEA	International Atomic Energy Agency
ILS	Iterated Local Search
LS2	Local Search 2
MOGA	Multi-Objective Genetic Algorithm
NFP	No Fit Polygon
OSP	Optimal Stacking Packing
PS	Pattern Search
PBP	Partial Bin Packing
RPV	Reactor Pressure Vessel
SA	Simulated Annealing
SPV	Smallest Position Value
STS	Stochastic Tree Search
TS	Tabu Search
VNS	Variable Neighbourhood Search

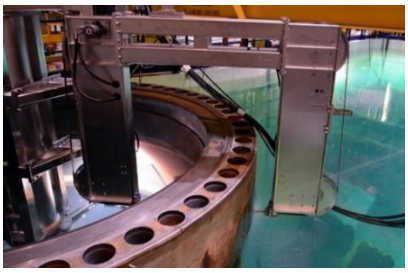
CHAPTER 1
INTRODUCTION

1.1 Background

Nuclear decommissioning is a complex and critical process involving the dismantling and safe disposal of materials from nuclear power plants that have reached the end of their operational life. According to the International Atomic Energy Agency (IAEA), as of 2020 there are a total of 686 reactors globally in various stages of their life cycles, with 442 being fully operational, 52 under construction, 172 permanently shut down and 20 that have been fully decommissioned [1]. In the UK alone, decommissioning activities are expected to generate waste until 2136, producing a total volume of approximately 4.45×10^6 m^3 [2]. Of particular relevance to this work are contaminated and activated metallic wastes (often arising from the size reduction of structures), which account for an estimated 436,200 m^3 of this total [2].

During the decommissioning of these facilities, it is inevitable that some structural components, such as reactor pressure vessels (RPVs), glove boxes, and pipe networks, will require size reduction via cutting before they can be packed for transport and disposal (e.g. Fig. 1-1, 1-2, & 1-3). The cutting process often involves specialised tools, such as band saws (Fig. 1-1 a), laser cutters (Fig. 1-1 b), electric hand saws (Fig. 1-2 a), plasma torches (Fig. 1-3 a) and mechanical shears (Fig. 1-3 b), while packing requires the placement of cut parts into containers for storage or transport (Fig. 1-1 c, 1-2 b, 1-3 c).

Both operations, however, come with significant financial costs that must be carefully considered. The costs associated with cutting and packing stem from several operational factors. Cutting costs include the purchase, operation, and maintenance costs of the cutting tools, along with labour costs for human workers or teleoperated robotic systems [3-5]. Packing costs primarily arise from the lifecycle expenses of containers, such as manufacturing, transportation, and storage costs, as well as labour costs for packing operations, whether performed by humans or teleoperated robots [6-8].



(a)

(b)

(c)

Fig. 1-1. RPV cutting and packing. (a) – bandsaw cutting of an RPV at Bohunice, Slovakia [9]. (b) – remote laser cutting of Dragon reactor core neck ring at Winfrith, UK [10]. (c) – Packed segments of an RPV from the Vermont Yankee reactor, US [11].

(a)

(b)

Fig. 1-2. Glovebox cutting and packing at the Institute for Reference Materials and Measurements, Belgium. (a) – electric hand saw cutting of a glovebox [12]. (b) – cut glovebox parts packed into a waste drum [12].

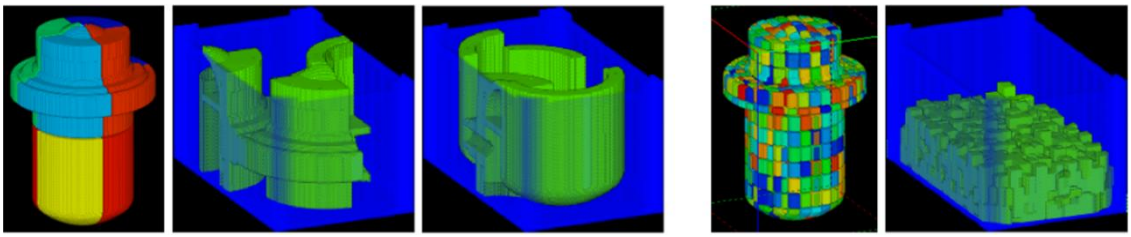
(a)

(b)

(c)

Fig. 1-3. Heat exchanger cutting and packing at the Horia Hulubei National Institute of Physics and Nuclear Engineering, Romania. (a) – manual plasma torch cutting of pipe segments [13]. (b) – remote operated mechanical shear cutting of pipe segments [13]. (c) – pipe segments packed into a waste drum [13].

The problem with cutting and packing costs are that they exist in a trade-off. Consider, for example, the structure depicted in Fig. 1-4. In the left case, the structure is cut using fewer cuts than in the right case, resulting in a much lower cutting cost. The trade-off however is that it produces larger cut parts which do not pack together well, resulting in two containers needed instead of one. It Is difficult to know, prior to performing the cutting and packing, which of the two solutions (if any) is more cost effective. Furthermore, it may be possible to cut the structure with fewer cuts than in the right case and still pack the pieces into a single container, potentially yielding a better solution than either approach.



(a) (b)

Fig. 1-4. Example of two ways to cut and pack a structure. (a) Few cuts (low cutting cost) and more containers (high packing cost). (b) Many cuts (high cutting cost) and fewer containers (low packing cost). Images courtesy of [14].

Despite its significance regarding optimal decommissioning strategies in nuclear decommissioning, this trade-off has received very limited attention in both academic and industry literature. To the best of the authors knowledge, the only case where this trade-off between cutting and packing has been experimentally analysed is in [15].

In this study, the authors examine the cutting and packing of Zircaloy cruciform elements from the fuel rod assemblies used in the Trino nuclear power plant in Italy. Four different dismantling solutions were proposed (A-D in Table 1-1), which were then physically tested and compared:

- **Solution A** involved horizontally cutting each element into three large sections (each approximately 100 cm in height) which were directly loaded into the containers without further processing.
- **Solution B** used the same horizontal cutting strategy as A, but with an additional compaction step to try and improve volume reduction.
- **Solution C** involved cutting at the gaps between two non-consecutive central stumps, allowing the cruciform element to be rearranged prior to packing.

- **Solution D** involved placing large vertical cuts along the overall height of the element, with the cut parts also being rearranged prior to packing.

Whilst solution A involved a simple cutting approach, it resulted in low volume reduction and a large number of containers, increasing packing cost. For solution B, the added compaction step improved volume reduction but caused the cruciform parts to shatter into many small fragments, significantly complicating their remote retrieval from the cutting space. Solution C offered a more balanced approach, achieving high volume reduction with a relatively low amount of cutting. Solution D maximised volume reduction via extensive cutting but introduced additional issues in the form of large secondary waste generation (metal swarf from milling) and significant vibration of the piece during cutting.

TABLE 1-1. Summary of dismantling solutions for Zircaloy cruciform elements. Table adapted from [15].

Solution	No. Cuts	No. Pieces	Total Cutting Length (cm)	Volume Reduction	Simplicity of Execution	No. of Machines Required
A	2	3	76.8	Low	Simple	1
B	2	3	76.8	High	Simple	2
C	3	8	114.6	High	Medium	1
D	3	12	376.8	High	Difficult	1

Based on their findings, solution C (shown in Fig. 1-5) was chosen as the optimum strategy. Although the authors did not compute a single scalar ‘total cost’ for the four options, their choice of C reflects an implicit total cost assessment: C delivered high volume reduction (fewer containers) without the extra equipment and handling difficulty of B or the secondary waste generation and vibration problems of D, and avoided the poor container utilisation of A, yielding the best overall balance between cutting effort and volume reduction.

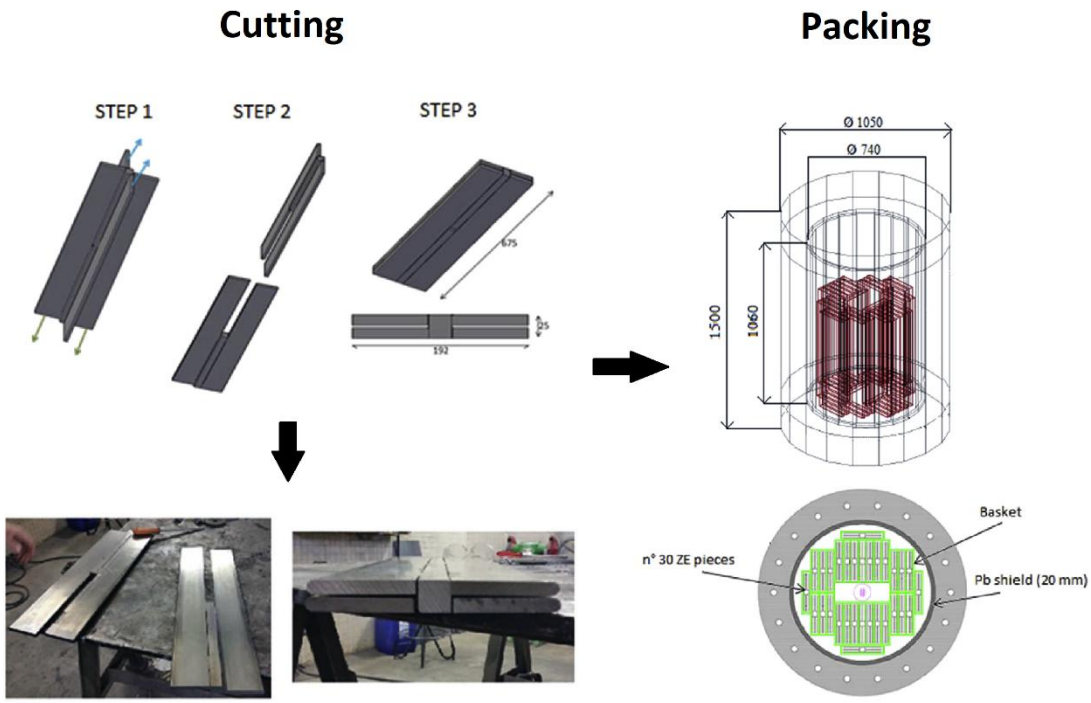


Fig. 1-5. Optimum cutting and packing strategy (solution C in Table 1-1) for Zircaloy cruciform elements. Images from [15].

Ultimately, the main challenge in the cutting and packing process lies in knowing where this optimal trade-off between cutting and packing exists. By increasing the size of cuts parts, it follows that the amount of cutting preformed is reduced (in turn lowering cutting costs), however this increase in cut part sizes will make it harder to find good packing arrangements (compact arrangements with minimal void space) which minimise the number of containers required. In contrast, cutting structures into smaller pieces helps improve packing density but will increase labour costs, tool usage and potential radiation exposure (for cases where human operators are used to perform cutting). Furthermore, the use of problem-specific tool and container combinations will heavily impact where this optimum trade-off lies, meaning there is no 'one-size-fits-all' solution which can be applied to all decommissioning scenarios.

In current practice, the problem of cutting and packing is often tackled manually. For large, high-value structures such as RPVs, detailed 3D models may already exist which can be used by engineers to manually plan cutting and packing strategies in advance (e.g. [16], [17]). For smaller or less standardised structures, such as the glovebox and heat exchanger pipes depicted in Figs. 1-2 and 1-3 respectively, the strategy is often left to the discretion of the operators working inside the shielded hot cells, who decide in situ how best to cut and pack the structure. These approaches, while practical, have important drawbacks: either

the process lacks optimality (as in manual in situ decision making), or, in the cases where a 3D model exists, identifying the best trade-off requires time-consuming manual planning and multiple iterations of trial and error. Furthermore, even after careful planning, it remains difficult to know whether a chosen strategy is close to optimal. This highlights the need for a more generalisable, automated approach that can explore a wide range of cutting and packing options for arbitrary structures, assess their trade-offs automatically, and identify optimal solutions without relying on physical trials or manual planning.

These practical limitations in current planning methods also introduce broader challenges in terms of accurate cost and resource estimation during the early stages of decommissioning planning. Because the optimum balance between cutting and packing is highly context dependent, even experienced engineers may find it difficult to reliably predict which strategy will lead to the most cost-effective outcome without extensive trial and error. This uncertainty can result in suboptimal decommissioning outcomes (e.g. unnecessary segmentation or inefficient container usage) and poor cost estimation during the planning phase (e.g. overestimating the number of containers required).

This is illustrated in the Zircaloy cruciform study ([15]) outlined above, where multiple cutting and packing strategies had to be trialled to evaluate trade-offs. Whilst this hands-on approach proved effective in the context of this small-scale study, their process required pre-defined strategies, relied heavily on human judgement, and lacked the ability to generalise across other structures. In contrast, the approach developed in this thesis aims to automate this process. By embedding both cutting and packing within a unified feedback-driven optimisation framework, the proposed algorithm aims to dynamically explore this trade-off and converge to cost-optimised solutions without requiring human-led iterative planning or physical trials. The result is a scalable and generalisable solution approach for arbitrary structures (under stated tool/container assumptions) that enables the planning of decommissioning tasks to be carried out more efficiently and accurately.

1.2 Project Aim and Objectives

The primary aim of this project is to develop a computational optimisation algorithm that can jointly consider cutting and packing for large irregular structures. The algorithm will take as an input a 3D model of a structure and will determine a cost-optimised cutting and packing strategy for it, explicitly addressing the trade-off between segmentation effort and container efficiency, whilst considering factors such as tool usage and container costs.

Objectives:

1. **Review literature and propose methodology** – Conduct a review on existing literature on cutting and packing with particular focus on how previous approaches attempt to link the cutting and packing problem and how packing algorithms developed for the packing of 3D irregular objects tackle the complex task of optimally arranging heterogeneous objects in 3D space to maximise packing efficiency. Use this to propose a combined methodology for optimising both processes in a unified framework.

2. **Develop a 2D prototype for the cutting and packing algorithm** – To simplify early implementation and accelerate testing, develop a 2D prototype of the cutting and packing algorithm. Use this simplified version to evaluate the proposed joint optimisation approach and identify areas for improvement before extending the methodology to the full 3D framework.

3. **Investigate methods to enhance solution quality and applicability** – Following the testing of the 2D prototype, investigate strategies to improve the underlying cutting and packing processes (in 2D) to enhance solution quality and real-world applicability. Use insights from this work to inform the development of the 3D optimisation framework.

4. **Extend the framework to 3D** – Update the methodology based on observations from the 2D testing, then translate the updated methodology into a 3D optimisation framework, capable of processing 3D input models of real-world structures and producing optimised cutting and packing plans for them.

5. **Test the 3D algorithm** – Demonstrate the algorithms ability to handle different decommissioning scenarios by testing it using different models of real-world structures and containers found in decommissioning. Benchmark its effectiveness against conventional strategies to cutting and packing, including ‘minimal-cut-then-pack’ approaches and manually generated cutting patterns. Perform sensitivity analyses to assess its robustness under varying cost assumptions and trade-off scenarios.

1.3 Thesis Structure

- **Chapter 1 – Introduction:** This chapter introduces the problem of finding optimal trade-offs in cutting and packing, highlighting the motivation for this research. Based on this, the aims and objectives for this project are defined.

- **Chapter 2 – Literature Review:** To provide a foundation for the proposed approach, this chapter reviews existing literature on cutting and packing problems. It is structured in three parts: the first section reviews the limited prior work attempting to link cutting and packing and is used to highlight the motivation behind the proposed integration strategy. The second section provides a detailed review of 3D irregular object packing algorithms, detailing how complex packing problems are approached in existing literature. The final section then discusses key considerations specific to cutting optimisation in nuclear decommissioning.

- **Chapter 3 – Methodology:** This chapter outlines the proposed methodology for solving the cutting and packing problem through an integrated optimisation framework. The approach is presented in a generalised form (which is applicable to both 2D and 3D representations) and serves as the foundation for the implementation and testing presented in later chapters. Details are given on the structure of the proposed framework, including input representation, cost modelling, and the overall optimisation strategy. Methodologies relating to the additional investigations into improving the algorithm are introduced in the chapters where they are implemented and tested.

- **Chapter 4 – 2D work:** This chapter covers the 2D work conducted in this project. The first part covers the 2D implementation and testing of the proposed cutting and packing algorithm, including an analysis of problems identified with the proposed methodology based on the results from testing. The remainder of the chapter focuses on investigating ways to improve the algorithm, including hyper-heuristic optimisation for improving container utilisation, comparing different strategies for multi-container packing and investigating ways to optimise disassembly sequences for cut structures whilst adhering to real world constraints (such as structural stability). Based on these studies, a detailed overview is given for potential areas of future work.

- **Chapter 5 – 3D work:** This chapter covers the 3D work conducted in this project. It begins by outlining modifications made to the proposed methodology based on findings from the work carried out in the 2D chapter. The implementation of the 3D algorithm is then described and a detailed outline of the test scenarios used to assess the algorithms performance is given. The final part of this chapter details the results from these tests, including the algorithm's performance on different structure and container combinations and benchmarking against conventional strategies such as minimal-cutting-followed-by-packing and manually planned cutting approaches. Based on these results, limitations with the existing algorithm are identified and areas for future work are discussed.

- **Chapter 6 – Conclusions:** This chapter summarises the key findings of the research presented in this thesis and reflects on the effectiveness of the developed cutting and packing algorithm. It outlines the main contributions of the project and highlights limitations with the current methodology, along with providing a summary of proposed areas for future work, as discussed in detail in the 2D and 3D work chapters.

Fig. 1-6 illustrates the structure of the thesis, outlining the tasks undertaken in this research project and how each task relates to the objectives outlined in the previous section.

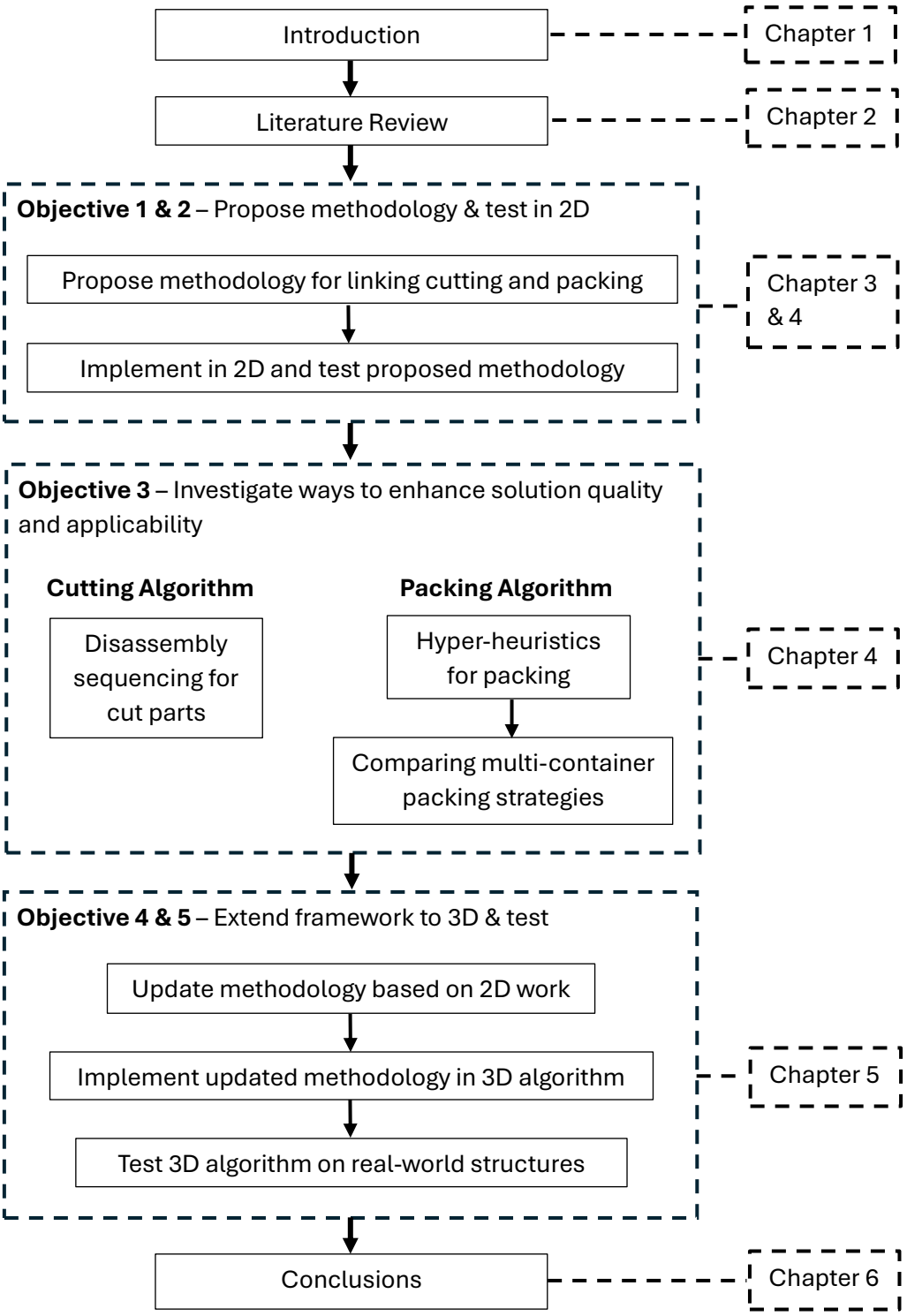


Fig. 1-6. Thesis structure and tasks.

CHAPTER 2
LITERATURE REVIEW

The current body of literature presents a very limited number of studies that directly address the trade-off between cutting and packing. Given the scarcity of research on this topic, this literature review will be structured in three parts. The first section will examine the few existing works that consider this dual optimisation problem, with particular focus on the methodologies employed to balance cutting and packing efficiencies as well as examining limitations which may make these approaches less suitable for nuclear decommissioning. The second section explores algorithms for 3D irregular object packing, including discussions on object representation, geometric complexity and its impact on computational demands, as well as the strategies used for placement optimisation and the constraints inherent in these approaches. The final section considers the specific challenges associated with optimising cutting processes in the context of nuclear decommissioning, highlighting areas of potential future research and innovation.

2.1 Review of Previous Approaches to Cutting and Packing

The primary issue with existing cutting optimisation [18], [19] and packing optimisation [20], [21] algorithms developed for nuclear decommissioning are that they fail to address the trade-off between cutting and packing.

The cutting algorithms in [18], [19] focus solely on minimising the amount of cutting required while ensuring all parts are within user-defined size limits, but they do not perform any kind of packing optimisation on the cut parts. As shown previously in Fig.1-4, when a structure is cut using a cutting-cost minimised solution, it will lead to large parts which may not pack together well (resulting in more containers required to pack them). By only seeking to minimise cutting (and ignoring packing), this will likely lead to cost inefficient solutions (with high packing cost) should a user wish to then pack the cut parts.

In contrast, the packing algorithms in [20], [21] only optimise packing for a fixed set of objects without accounting for the cutting process. In [21], they start with a pre-partitioned 3D model of an RPV whereas in [20] they use 3D point clouds obtained by scanning real world waste objects. However, none of the algorithms consider how cutting (e.g. modifying the initial partitioning in [21], or splitting the scanned objects into smaller pieces in [20]) would affect the algorithms ability to pack the parts. Consequently, if a user wished to

obtain cost-optimised cutting and packing solutions using an existing packing algorithm, the user would have to manually design multiple different cutting patterns, re-running packing optimisation on each one. The problem with this approach is that it would be time consuming and requires a high degree of human input.

To date, there are very few examples in literature where this trade-off between cutting and packing has been considered, with most examples being developed for additive manufacturing [22-25]. The goal of these algorithms is to partition an input object (which may be larger than the printing volume) and then pack the pieces into the printing volume with the goal being to optimise:

1. Printing time/support material – finding packing solutions with part orientations/locations which facilitate faster printing and minimise the amount of support material required for printing overhangs.
2. Packing volume – minimising the space required to pack the printed parts, useful for commercial cases where, for example, multiple instances of an object need to be printed and shipped in boxes.
3. Constraint based partitioning – optimising the partitioning of the structure to meet problem specific objectives such as ease of assembly, avoiding cuts through high stress regions, and minimising the number of parts.

The problem with approaches [22], [23] and [24] is their heavy reliance on a good initial partitioning. In [22], the authors solve the cutting and packing problem in an iterative fashion, starting with an initial partition and packing solution, and then iteratively optimising the partitioning (to enforce constraints such as minimising cuts through high stress regions – Fig. 2-1) and modifying the packing solution (to remove any overlap between parts caused by the modification of the partitioning). Whilst they do allow cuts to be modified during the optimisation process, they do not allow cuts to be removed or added (i.e. meaning the number of cut parts will always stay the same). They do offer the user several options in terms of initial partitioning schemes (e.g. shape diameter, random walk, random cuts, normalised cuts, fitting primitives, k-means and core extraction [22]), however this still limits the flexibility of the initial partitioning to a fixed set of options, and also requires the user to manually re-run the algorithm with the different initial partitioning options.

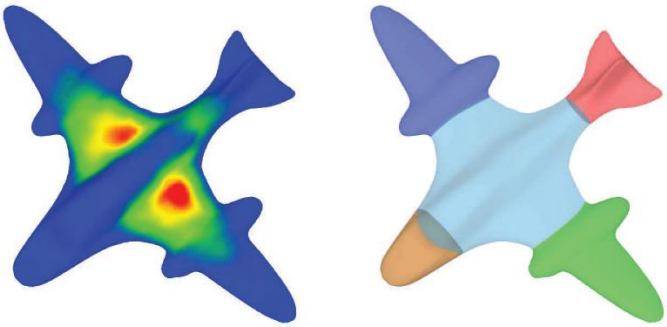


Fig. 2-1. Example of stress-field analysis from [22], used to avoid placing cuts through high-stress regions.

In [23], the authors pack the initially partitioned object, after which the algorithm enters a cycle of trying to remove cuts (by merging parts together) and then re-packing them. They do not however allow existing cuts to be modified, or new cuts to be added, during this process. To generate the initial cutting pattern, they partition the mesh into N clusters of roughly equal volume (where N is set by the user) and then perform a pre-merging step to merge certain parts based on problem specific criteria (e.g. avoiding cuts through small cross sectional areas of the model [23]). The issue here again is that if a user wished to try different initial decompositions, they would need to manually re-run the algorithm for different values of N. Furthermore, the method's constraint of generating partitions with approximately equal volume reduces the flexibility of the initial partitioning, potentially limiting its effectiveness in scenarios where non-uniform partitioning may be more suitable.

An additional point worth noting with this approach is their use of hollow shells to represent objects (Fig. 2-2). Before the initial segmentation stage, the input mesh model is hollowed out to remove the internal volume, leaving a 3D shell version of the input model. With regards to additive manufacturing, the benefit of this is that: a.) it greatly reduces the amount of material required to print the structure (since the internal material is removed), and b.) it allows the hollow parts to fit together well during the packing, allowing for tighter packing arrangements. Naturally, such an approach would be unsuitable for nuclear decommissioning since the parts would be cut from an existing structure rather than being printed from scratch. As such, it would be wholly unsuitable to remove internal volume from the input model as it would no longer be an accurate representation of the real-world structure.

Fig. 2-2. Example of hollow shell decomposition from [23].

In [24] cuts can only be added during the packing phase (where objects are packed one by one, with parts sometimes being split to facilitate more compact packing), but they do not allow existing cuts to be removed or modified. They generate the initial partitioning by decomposing the structure into a small number of 'pyramidal primitive parts' [24]. The use of pyramidal primitives makes sense within the context of additive manufacturing as it produces parts which can be printed in their upright orientations with no support material (e.g. Fig. 2-3) [24]. For nuclear decommissioning however, again, this limits the flexibility of the cutting approach by restricting the initial partitioning to a set type of cut parts (i.e. pyramidal primitives).

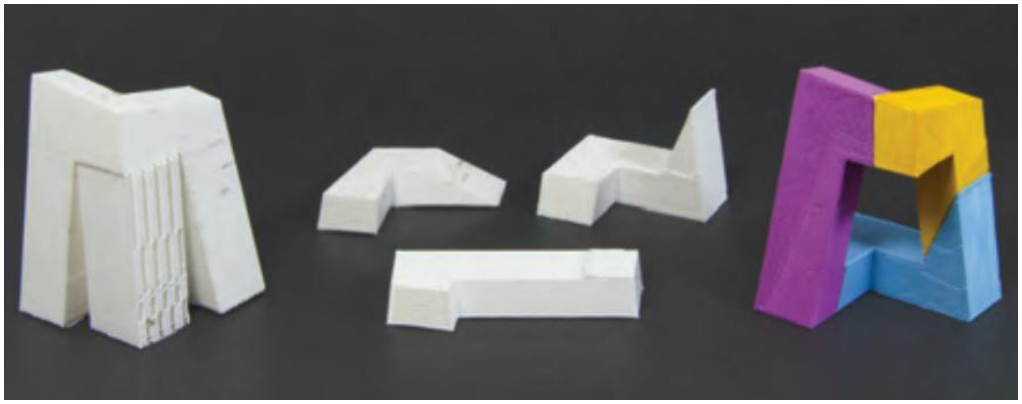


Fig. 2-3. Example of whole printed structure with support material (left), and pyramidal decomposition allowing parts to be printed without support material (right). Images from [26].

In [22], the authors demonstrate their algorithm with the different initial partitioning schemes available to the user, and show that the quality of the solutions (in particular the packing space utilisation ratio) varies depending on the initial partitioning scheme. The primary difficulty with the simultaneous optimisation of cutting and packing lies in the difficulty of knowing what a 'good' initial partitioning will be (i.e. a partitioning which leads

to maximal packing space utilisation with minimal cutting). The problem of initialisation is arguably of less concern in additive manufacturing since these algorithms focus more on generating 'good enough' solutions in a short space of time, rather than spending a long time trying to improve the quality of a solution by a few percent. However, in the context of nuclear decommissioning, it can be argued that the latter is of more importance. Due to the technological and safety challenges associated with these unique environments, planning decommissioning strategies is a lengthy process often taking many months. It is not uncommon for the entire decommissioning process (i.e. planning, decommissioning and site remediation) to take years to complete [27]. Furthermore, nuclear decommissioning is a very expensive process, where improvements of even a few percent could translate to millions in financial cost savings. Given both factors, it can therefore be argued that solution quality (i.e. minimisation of cost) is of far greater priority when compared to speed.

One possible solution to this problem would be to run the algorithms in a looped fashion, each time running it with a different initial decomposition, but none of them do this (at least not in an automated fashion). Additionally, this would still not alleviate the problem of deciding how to change the initial partition in each loop. To truly help resolve this issue, the algorithm would need to consider information from both the cutting *and* packing solution produced by an initial partitioning to judge how good the starting decomposition is. The algorithm would need to use this information to try and guide the modification of the partitioning such that it leads to solutions which optimise both the cutting and the packing. The obvious issue with such an approach would be the increase in computation time, since the processes would have to run multiple times in a loop. However, again, it can be argued that such a trade-off would be acceptable if it leads to better optimised solutions.

An additional point to note with approaches [22] and [24] is that they do not re-run packing optimisation from the start each time the partitioning is modified. For example, in [22], once the algorithm has generated the initial cutting/packing solution and enters the iterative optimisation phase, any subsequent modifications to the cuts do not trigger a full re-packing of the parts. Instead, only local adjustments are made to the existing packing configuration to resolve part intersections introduced by the modification of the initial partitioning. In [24], once the initial partitioning is generated and the algorithm proceeds to the packing stage, parts are packed sequentially into the container. At each step, the algorithm has the option to either pack a part whole or to split it into two sub-parts before packing both. While this strategy can result in more locally compact configurations, it shares a similar limitation with [22] in that it does not re-pack the full set of parts from scratch when the partitioning is modified. A key limitation of not re-packing all parts from

scratch each time the partitioning changes is that changes to the cuts will inherently alter the geometric properties of the resulting parts. This in turn modifies the optimisation landscape of the packing problem, shifting both global and local optima. By focusing solely on local adjustments to an existing packing structure, rather than re-running packing optimisation each time the parts change, the algorithm risks overlooking more optimal packing arrangements that may now be achievable with the new set of parts.

The algorithm presented in [25] doesn't suffer from these problems, instead starting with a complete structure, which is then recursively split and re-packed until either the target packing efficiency, or maximum number of allowed parts, is reached. The key limitation with this approach lies in its requirement for the user to specify the target packing efficiency and maximum number of cut parts, with the quality of solutions being sensitive to these parameters. Additionally, if the algorithm reaches the part limit before achieving the target packing efficiency, the algorithm will become stuck, prompting the user to increase the maximum allowed number of parts [25]. From a nuclear decommissioning perspective, the issue here is that this is exactly the problem which needs to be solved, since it is not possible to know a priori what good values would be. For example, setting both the target packing efficiency and maximum number of parts too high would likely result in excessive splitting and re-packing, leading to solutions with good packing (low packing cost), but excessive cutting (high cutting cost).

Another limitation with all these approaches is that they only pack objects into a single container, and assume the container is a cuboid in each case. Again, this may not be problematic for additive manufacturing (assuming the printing space is perfectly cuboid and the model is small enough that the parts can all fit), however nuclear decommissioning often deals with large and complex structures (which require multiple containers to pack), and often utilises different container designs depending on factors like the radioactivity levels of the waste or whether the waste is compactable or not. Any algorithm developed for nuclear applications should have the ability to pack objects across multiple containers (whilst seeking to minimise the number of containers) and should also allow the use of different container types depending on the problem.

A more recent publication worth noting is the work in [28]. In this paper the authors present a cutting and packing algorithm designed specifically for RPVs in nuclear decommissioning. The problem is addressed in a feedback-loop fashion, with packing optimisation being run on each set of cut parts before the cutting/packing solutions are evaluated (based on the cutting time and container space utilisation). Additionally, a population-based metaheuristic (Genetic Algorithm) is used to optimise the partitioning, which helps alleviate

the dependence on a good initial partitioning by allowing the algorithm to start with a population of different cutting solutions.

Conceptually, their approach shares several similarities with the methodology presented in this thesis, including the repeated evaluation of packing solutions for candidate cutting patterns and the use of population-based metaheuristics to explore alternative partitioning strategies. A key distinction, however, is that the method in [28] formulates cutting and packing as a multi-objective optimisation problem, whereas the 3D cutting and packing algorithm presented in this thesis combines cutting and packing costs into a single weighted objective function.

One of the main strengths of the algorithm presented in [28] is the computational efficiency of its packing procedure. To achieve this, the authors simplify the 3D packing problem to a 2D one by projecting the 3D cut blocks onto a 2D plane (Fig. 2-4). By simplifying to 2D, it removes a spatial dimension from the problem which in turn reduces the search space (allowing solutions to be obtained faster). It also allows them to utilise a common and fast approach in 2D packing called the No Fit Polygon [29], which can be used to find collision-free locations for parts far quicker than conventional geometric approaches (which often require repeatedly testing for intersections between packed parts).

A key limitation with this approach however (as acknowledged by the authors), is that it is only applicable for packing cases where the height of the cut parts is uniform (or with negligible difference), such as when cutting an RPV into segments of uniform height. As such, their packing methodology would be unsuitable for packing arbitrary shapes, limiting its flexibility to deal with arbitrary structures.

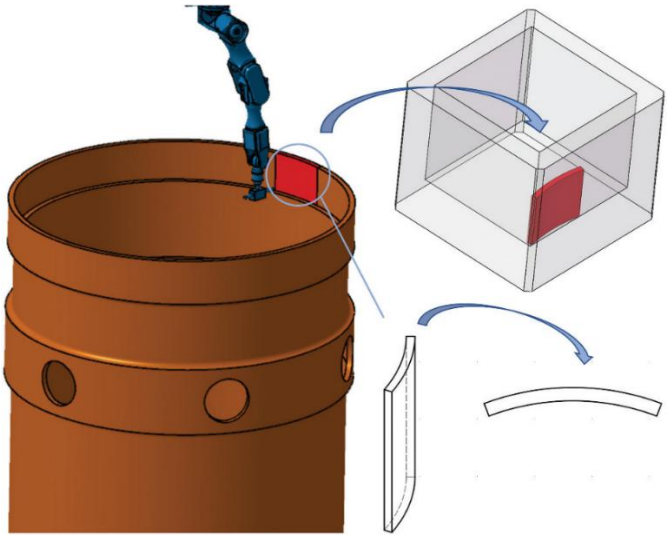


Fig. 2-4. Illustration of 3D cut blocks projected onto 2D plane from [28].

Another limitation of their work is that, as with the previous four algorithms, it also does not contain multi-container packing, instead assuming all parts will fit into a single large container. They also do not have the ability to use arbitrary container types, instead restricting the packing to cuboid containers. A final point to note is that the cutting approach they use is tailored specifically for cutting RPVs (by placing cuts radially from the central axis of symmetry). Whilst the cutting and packing methodology itself could be adapted to other structures, in its current form, their algorithm can only cut and pack RPV's, limiting its flexibility.

To the best of the author's knowledge, their work and the work presented in this thesis are the only two examples in literature where this trade-off between cutting and packing has been considered within a nuclear decommissioning context.

2.1.1 Concluding Remarks: Review of Previous Approaches to Cutting and Packing

This review of existing algorithms which have attempted to link the cutting and packing problem highlights several limitations which make them unsuitable for nuclear decommissioning. A major issue across several methods ([22], [23] and [24]) is their heavy reliance on the initial partitioning of the input object, with limited flexibility to modify the cutting patterns during optimisation. The method in [25] does not suffer from this issue, but requires the user to specify target packing efficiencies and maximum number of cut parts in advance, which is problematic for decommissioning since such values are not known a priori.

In addition to this, several approaches ([22], [24]) do not re-run packing from the start each time a cutting pattern is modified, which introduces the risk of overlooking better packing configurations. Although the method developed specifically for nuclear decommissioning ([28]) does address this by re-optimising packing for each new cutting pattern, its simplification of the packing problem limits it broader applicability to cutting and packing arbitrary structures.

A further limitation shared by all the reviewed methods is that they do not have the ability to pack objects across multiple containers and restrict packing to cuboid containers.

Table 2-1 below summarises the main limitations highlighted with each approach. These limitations highlight the need for a more adaptable framework, capable of handling arbitrary geometries, diverse container types and multi-container packing. This motivates the approach proposed in this thesis, which explicitly seeks to address these limitations in a generalisable algorithmic framework.

TABLE 2-1. Summary of limitations for the 5 cutting and packing algorithms outlined in this review.

Source – Year	Application	Limitations
[22] - 2015	Additive Manufacturing	Sensitive to initialisation. No repacking of parts when cuts change. Single (cuboid) container packing. Number of cut parts remains constant throughout.
[23] - 2014	Additive Manufacturing	Sensitive to initialisation. Single (cuboid) container packing. Cuts cannot be added or altered (only removed to merge parts). Input model converted to hollow shell.
[24] - 2015	Additive Manufacturing	Sensitive to initialisation. No repacking of parts when cuts change. Single (cuboid) container packing. Cuts can be added but not removed or altered.
[25] - 2015	Additive Manufacturing	Requires user to specify target packing efficiency and maximum number of parts. Single (cuboid) container packing.
[28] - 2025	Nuclear Decommissioning	Tailored specifically for RPV cutting and packing. Single (cuboid) container packing. Packing problem simplified from 3D to 2D (limits application to arbitrary shape packing).

2.2 Review of 3D irregular Object Packing Algorithms

Based on the review on previous approaches to cutting and packing optimisation presented in Section 2.1, it is evident that packing optimisation will play a crucial role in this project. Given that nuclear decommissioning often involves the cutting and packing of numerous different types of structures with varying levels of complexity, this review chapter will focus primarily on the packing optimisation of 3D irregular objects. The goal of 3D irregular object packing is to find an arrangement for a set of objects in 3D space, such that their compactness (i.e. how densely the objects are packed together) is maximised subject to no overlap between the packed parts.

This is typically achieved by one of three strategies:

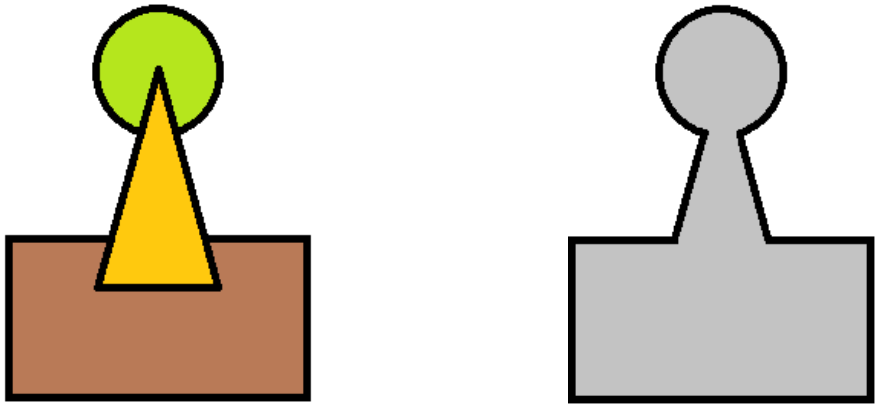
1. **Minimising bounding enclosure:** Minimising the convex hull or bounding box enclosure surrounding a set of objects in 3D space (e.g. [22], [30]). By compacting a set of objects, it follows that the bounding enclosure of the set of objects will become smaller.
2. **Minimising one dimension of an unbounded container:** Using a container with 2 fixed dimensions (e.g. the length and width) and one dimension (e.g. the height) set to infinite, where the goal is to then minimise the height of the packing pile (e.g. [24], [31]). Given that two dimensions are fixed, it follows that by minimising the unbounded dimension of the packed pile (e.g. the overall height of the packing structure) the compactness of the packed structure will increase.
3. **Maximising utilisation in fixed-size containers with possible subset selection:** When using a container with fixed dimensions, it is often infeasible to fit all objects into the container (which is especially true in multi container packing). In such cases, the problem involves an additional layer of complexity in the selection of subsets to pack into the container (e.g. [21], [32]). In such cases, the goal is to find a subset selection which maximises the amount of packed volume in the container, or, in the case of multi container packing, to partition the object set into multiple subsets, which lead to a reduction in the number of containers required to contain all the parts.

Whilst the method of compaction and optimisation objectives may differ from problem to problem, the constant factor among them is that they all try to maximise packing compactness. This review will focus primarily on the techniques they use to achieve this. In addition to this, critical factors relating to the real-world packing of waste objects, including the geometric complexity and representation of shapes (and how this impacts computational complexity), and practical constraints that must be considered in real-world waste packing scenarios (such as the stability of packed parts within the container), will also be discussed.

When it comes to packing optimisation algorithms, broadly speaking, they may be classified as either:

1. **Mathematical approaches**: which aim to formulate the packing problem as a mathematical optimisation problem which is guaranteed to contain the global optimum.
2. **Heuristic approaches**: which employ rule based, or metaheuristic strategies which sacrifice global optimality in favour of faster and more efficient exploration of the search space.

To date, the only methodology in packing literature which can formulate the packing problem mathematically is the 'phi-function' approach ([33-44]). The phi-function approach works by representing objects as either 'primary phi-objects' or 'composed phi-objects'. A primary phi-object is either a sphere, parallelepiped, circular cylinder, circular cone, or convex polyhedron [35]. Through the union or intersection of primary phi-objects, more complex objects (including concave objects) can be represented (referred to as 'composed' phi-objects – e.g. Fig.2-5 below).



(a) (b)

Fig. 2-5. 2D example of primary phi-objects forming a composed phi-object. (a) – 3 primary phi-objects (circle, triangle, rectangle). (b) – Union of primary phi-objects forming a composed phi-object.

Each primary phi-object has an associated phi-function which provides an analytic description of the object's geometry. Through the pairwise combination of primary object's phi functions, an analytic description of the distance between the objects can be formulated, which in turn can be used to check for object intersection. Consider, for example, the simple case illustrated in Fig. 2-6 for two circles. Circle 1 has a radius of r_1 and has its position fixed at the origin $(0,0)$. Circle 2 has a radius of r_2 and is allowed to move freely. The equation $\sqrt{x^2+y^2} = r_1 + r_2$ can be used to describe all possible positions of circle 2 such that both circles touch but do not overlap. It follows then that, $\sqrt{x^2+y^2} > r_1 + r_2$, when the circles are separated, and $\sqrt{x^2+y^2} < r_1 + r_2$ when the circles are intersecting. Hence, the phi-function describing the pairwise relationship between the two circles can be expressed as $\Phi(x,y) = \sqrt{x^2+y^2} - (r_1 + r_2)$, where:

- $\Phi(x,y) = 0$ when the shapes are touching,
- $\Phi(x,y) > 0$ when the shapes are apart,
- $\Phi(x,y) < 0$ when the shapes are intersecting.

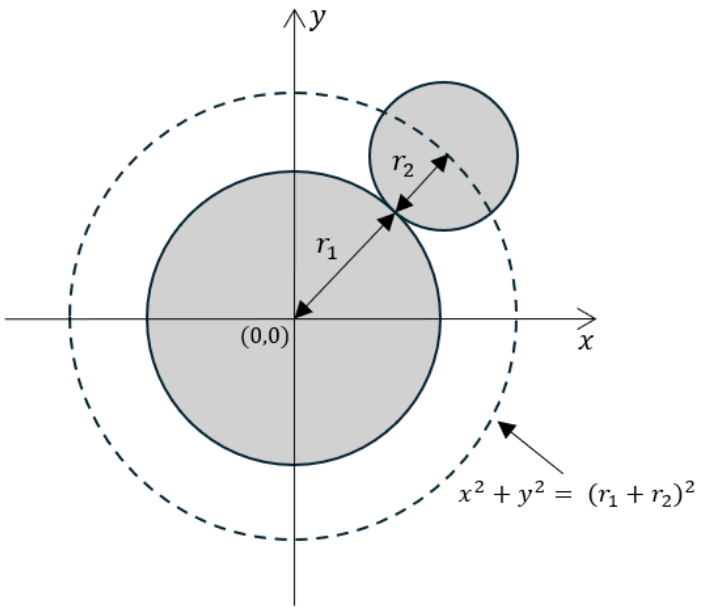


Fig. 2-6. Simple example of two primary phi-objects (circles) in contact with one another. Dotted line indicates all possible positions of circle 2 where the two circles are in contact.

It is worth noting that the case of two circles is the simplest possible case as the phi-function can be represented by a single function. However for more complex primary phi-objects, a series of functions must be derived before the object's phi-function can be obtained [45]. More detail on the construction of phi-functions for different shapes is given in [35].

From an optimisation perspective, the benefit of using phi-functions is that they provide an analytic description of the distance relationship between objects, considering the objects *continuous* rotation and translation. Employing this approach allows the packing problem to be formulated as a nonconvex and continuous nonlinear programming problem [33], [35], which can in theory be solved to global optimality using a nonlinear programming solver given enough time.

Whilst the phi-function approach represents a good first attempt at representing the 3D packing problem mathematically, there are several drawbacks which would limit its efficacy for the application of nuclear waste packing. These drawbacks are as follows:

1. **$O(n^2)$ pairwise intersection checking:** For a system with n objects, the computational complexity of pairwise overlap checking scales approximately with $O(n^2)$. Since complex objects are often composed of multiple phi-objects (each requiring their own phi-functions), the number of phi-function evaluations per object can grow significantly with an increase in both their number and complexity. Consequently, whilst the number of pairwise comparisons is quadratic with n, the computational cost of phi-function

evaluations can be much higher depending on the complexity of the objects involved. To try and reduce this issue, the authors of [46] introduce the concept of 'quasi-phi-functions', a variation of the phi-function including auxiliary variables, which is substantially simpler than conventional phi-functions for certain types of primitive shapes (such as ellipses and polyhedra). Whilst this does help reduce the computational burden associated with the intersection checking of such objects, it does not remove the problem of having to decompose complex objects into many primary phi-objects and it does not remove the need to perform pairwise checking.

2. **Sensitivity to initialisation:** The benefit of this mathematical formulation is that it allows all objects to be rotated and translated simultaneously and in a continuous fashion within the packing space. As stated, whilst this does theoretically mean the packing formulation is guaranteed to contain the globally optimum packing result, it does not guarantee this result will be found. In practice, the final optimised positions of the objects will depend heavily on their initial locations, meaning that the algorithm is only able to generate locally optimal solutions. In [33], the authors present a fast starting point algorithm to generate a feasible initial packing structure which is then followed by an optimisation compaction procedure (called COMPOLY). The COMPOLY procedure reduces the problem to a sequence of smaller dimension subproblems (with fewer non-linear inequalities) which are then solved by a non-linear programming solver [33]. However, even with their use of a fast starting-point algorithm, simplification procedure and use of quasi-phi-functions, the algorithm is still only able to generate locally optimal solutions, which take a long time to obtain. As an example, in [33], the authors present a solution to a packing problem (involving 80 convex polyhedra) originally presented in [40]. Using their fast starting point and COMPOLY procedure, they are able to achieve an improvement of 27.88% in the packing solution compared to [40], however the reported run time to achieve this was 46035.78 seconds (≈12.8 hours).

3. **Decomposition of complex objects:** As stated, the problem with representing complex 3D objects is that they need to be decomposed into multiple primary phi-objects. Currently no algorithm exists which can automatically decompose arbitrary 3D shapes into primary phi-objects and calculate the corresponding phi-functions [45]. Consequently, the application of phi-function methods in literature is largely restricted to relatively simple and predominantly homogeneous shapes. This limitation remains a significant open challenge in the application of phi-function based packing approaches to complex real-world packing problems.

Due to the problems associated with mathematical approaches (specifically, computational complexity and difficulty with representing arbitrary shapes), this approach, at present, remains unsuitable for the application of nuclear waste packing (which can often include many complex and highly heterogeneous objects depending on the complexity of the problem). As such, attention is turned toward heuristic packing approaches for the remainder of this review.

As stated, heuristic techniques are more problem dependent approaches which aim to find good locally optimal solutions in shorter time spans but have no guarantees of global optimality. Heuristic packing approaches may further be split into two separate classes; constructive heuristics and metaheuristics. Constructive heuristics are low-level heuristic approaches which incrementally build a solution from an initially empty solution, stopping once a complete solution is found. In contrast, metaheuristic approaches are constructive approaches which incorporate metaheuristic search techniques (e.g. genetic algorithms) to help escape local optima and find better solutions.

In the following sections, relevant literature on constructive heuristic and metaheuristic approaches to 3D irregular object packing problems will be presented and discussed in greater detail (Sections 2.2.1 and 2.2.2 respectively). Additionally, a discussion on aspects relating to the design and implementation of packing algorithms and how these relate to real world packing of waste will also be presented (Section 2.2.3). Specifically, the following points will be discussed:

- **Object Representation** – how objects are represented in different packing approaches and how this impacts computational complexity as well as an algorithm's ability to capture complex real-world geometry.
- **Object Complexity** – how the geometric complexity of objects (i.e. how 'irregular' the objects are) can affect solution approaches and algorithmic complexity.
- **Container Type** – focusing specifically on whether algorithms use single container or multi container packing and whether they use bounded containers (containers with fixed dimensions) or unbounded containers (containers with one, or multiple dimensions unbounded).
- **Constraints** – types of constraints used in previous packing algorithms with particular focus on whether any approaches from previous literature address constraint requirements (like stable and physically accessible part locations) in real world packing problems.

- **Rotation** – examining how algorithms deal with object rotation and how this affects computational complexity.

2.2.1 Constructive Heuristics

Constructive heuristics represent a class of intuitive, rule-based methods for incrementally building solutions from the ground up. Rather than searching over the entire solution space, these approaches sacrifice optimality in favour of generating approximate solutions more quickly. Within the context of 3D irregular object packing optimisation, a variety of tailored constructive strategies have been proposed, each designed to accommodate geometric complexity and spatial constraints related to the problems they are applied to. Table 2-2 below provides a summary of publications where constructive heuristics have been used for 3D irregular packing, highlighting the diversity of strategies developed to address this class of problem.

TABLE 2-2. Publications based on constructive heuristic approaches.

Source - (year)	Object Representation	Object Complexity	Container Type	Packing Method	Constraints	Rotation Scheme
[47] - 2001	Voxel	High	Single container (bounded)	Objects dropped from top of container (addition rate set by user). Biased random walk which favours downward motion.	No overlap. Containment.	'on' (1-degree increments), or 'off' (no rotation).
[30] - 2008	Mesh	Med.	Single container (unbounded)	Objects start spread out. 'Rubber band' placed around objects which draws them together under elastic forces.	No overlap.	Continuous (objects only rotate if in contact with other objects).
[48] - 2009	Voxel	High	Single container (bounded)	Speed optimised version of [47]. Biased random walk.	No overlap. Containment.	Finite (set by user)
[49] - 2010	Mesh	Med.	Single container (bounded)	3 stage process; large objects grouped and packed one by one using 'quad-wall' placement heuristic. Medium objects placed one by one (largest vol. to smallest) using bottom-left-back fill. Remaining small items packed using wall building.	No overlap. Containment. Stability. Problem specific.	Finite (90°)
[50] - 2013	Cluster of parallelepipeds	Med.	Single container (bounded)	Problem formulated as Mixed Integer Nonlinear Programme. 'Approximate' initial layout generated with some overlap allowed. MINLP solver optimises positions/rotations of	No overlap. Containment.	Finite (90°)

				objects simultaneously to minimise packing height.		
[51] - 2014	Cuboid approximation	Low	Single container (bounded)	One by one packing approach using Bottom-left-back-fill. Similar objects first grouped together. Sequencing algorithm calculates packing order and poses for part groups and parts within each group. Sequence aims for largest-to-smallest packing order but can change if a part fails to pack.	No overlap. Containment.	Finite (90°)
[24] - 2015	Voxel (during packing). Mesh (during local refinement)	High	Single container (bounded)	Input model partitioned into pyramidal primitives which then voxelised. Voxel objects packed one by one (bounded beam search selects which object to add next). Objects can also be split to facilitate tighter packing. Final refinement step to improve compaction. In both stages, height and vertical gaps are minimised.	No overlap. Containment.	Finite (90°) in voxel packing. Continuous in local refinement.
[22] - 2015	Level-set functions	High	Single container (unbounded)	Input model partitioned; parts converted to level-set representation. For packing, parts start spread out, container then shrunk and parts moved inside new boundary resulting in overlap. Parts iteratively translated and rotated to remove overlap. If overlap can't be removed, level-set functions adjusted to remove overlap.	No overlap. Containment. Problem specific.	Continuous (can be manually restricted by user)
[25] - 2015	Voxel	High	Single container (bounded)	One by one placement (largest to smallest) using hole filling and height minimisation approach. Parts can be split further to facilitate better packing.	No overlap. Containment.	Finite (10 and 40 uniformly spaced rotations)
[52] - 2015	Mesh	Low	Single container (unbounded)	One-by-one placement (largest vole to smallest) to minimise centre height of structure.	No overlap. Containment.	Finite (45°)
[53] - 2016	Voxel	High	Single container (unbounded)	One by one placement (largest vol. to smallest) using height minimisation. 3D adaptation of No Fit Polygon used to find locations.	No overlap. Containment.	None
[54] - 2018	Mesh	High	Single container (bounded)	Packing space partitioned into cells, objects placed in cells at 10% their original size. Algorithm then tries to 'grow' objects to their original size. If an object can't fully grow, it can be swapped with other objects or replaced with non-packed objects.	No overlap. Containment.	Finite (20 orientations with minor perturbation allowed)
[55] - 2018	Mesh	High	Single container (bounded)	One by one placement (largest vol. to smallest or user specified order) using heightmap minimisation. Placement locations must be accessible by robotic manipulator and allow objects to remain in static equilibrium.	No overlap. Containment. Robot packable. Stability.	Finite (rotation granularity of 45° or 22.5° with N randomly sampled points)
[20] - 2019	Point cloud (points as spheres)	High	Single container (unbounded)	Objects packed sequentially (largest vol. to smallest) by aligning and	No overlap.	Finite (90° about first principal

				positioning point clouds to maximise contact area.		component axis)
[56] - 2019	Mesh	Low	Single container (unbounded)	Objects start randomly dispersed. Objects drawn together using dynamic vector fields to simulate motion/rotation.	No overlap.	Continuous
[57] - 2020	Cluster of parallelepipeds	High	Single container (bounded)	One by one placement (prioritises large objects first). For each object, placement heuristic identifies 'free regions' in the container which can contain it. Goal is to select region/object orientation combination which minimises unused space.	No overlap. Containment.	Finite (90°)
[58] - 2023	Voxel (during packing). Mesh (during local refinement)	High	Single & multi container (bounded)	One by one placement (largest vol. to smallest) using height minimisation penalty function. Local refinement step to improve compaction. First fit decreasing used for multi container packing.	No overlap. Containment. Interlock free.	Finite (90° in examples)
[59] - 2024	Voxel	High	Single container (bounded & unbounded)	One by one placement (largest vol. to smallest) using deepest bottom left fill. Followed by shaking process to improve compaction (object motion modelled using rigid body dynamics).	No overlap. Containment.	Finite during initialisation (90°), continuous during shaking.
[60] - 2024	Mesh	Med.	Single container (unbounded)	Objects start spread out. Objects drawn together by attraction-based acceleration. The farther apart two objects are, the stronger the attraction between them.	No overlap. Problem specific.	Continuous

When it comes to constructive approaches to packing, the most common approach is one-by-one placement, where objects are placed one-by-one into the packing space (in a given packing order), with the location of objects determined by a 'placement heuristic' (or placement 'rule'). For example, in [55], the authors pack objects into the container one-by-one (with objects ordered from largest volume to smallest and packed starting with the largest), selecting object locations using a novel 'heightmap minimisation' placement heuristic, which aims to place objects as low as possible in the packing structure to minimise the increase in the packing structure's heightmap. In [58], the authors use a similar one-by-one placement approach to [55], which also packs objects from largest volume to smallest, and aims to minimise the height of the packing structure. They also include an additional post-processing step, which aims to compact the structure further by minimising the spacing between packed objects.

In some cases, rather than placing all objects individually, authors may also employ grouping procedures to group subsets of items together before packing to minimise the

number of variables (i.e. the number of objects to pack). For example, In [51], the authors pack objects one by one using a bottom-left-back-fill placement heuristic (a 3D adaptation of the popular bottom-left placement heuristic often used in 2D packing), which aims to place objects as close to the bottom-left-back corner of the container as possible. Before packing, they use a pre-processing step which groups objects based on the similarity of their geometric properties. In [49] the authors present a 3 stage packing approach for loading furniture into a van. The algorithm first groups large objects together which are packed into the van using a 'quad-wall' approach, which aims to select a grouping of 4 large subsets such that they can fit next to each other in the width and height of the container (to form a 'quad-wall') [49]. Medium sized objects are then packed one-by-one (largest volume to smallest) using a bottom-left-back-fill approach. Finally, the smallest items are packed into the remaining available space at the end of the van using a 'wall-building' approach (which aims to stack object on top of one another to form 'walls').

The primary benefit of one-by-one placement approaches is that they mirror how humans or robotic systems pack objects in practice. Since the packing sequence is explicitly defined in the problem, humans (or robotic systems) can easily replicate the packing sequence in real life. Furthermore, the use of one-by-one placement also makes the incorporation of domain specific constraints (such as object stability and location accessibility) much easier since these constraints can easily be checked for violation each time objects are added.

The main drawbacks with this approach however are that the choice of placement heuristic and packing order can significantly affect the quality of the packing result. To help mitigate the issue of packing order, one-by-one approaches are often combined with metaheuristic search techniques (such as genetic algorithms) to optimise the packing order (e.g. in [61-63]). Another technique, more rarely used, is the concept of 'hyper-heuristics', where rather than having access to only a single placement heuristic, the algorithm may select from a set of placement heuristics. To the best of the authors knowledge, the only example in literature where this technique has been employed in 3D irregular object packing is in [31]. In this paper, the authors present several metaheuristic algorithms for solving 3D irregular packing problems, including one algorithm where the packing order and placement heuristics are optimised using iterated local search. As a point to note, for constructive approaches where the packing order is fixed, the objects are typically sorted from largest volume to smallest and packed in this order starting with the largest. The reasoning behind this is that it will lead to large objects being placed first (in the bottom of the container), with smaller objects filling the gaps between them as they are subsequently added.

In contrast to one-by-one placement (which aims to pack objects in a serial fashion), another common approach is to solve the problem in a parallel fashion, starting with an initial layout of all the objects in the container, and then translating/rotating the objects (either simultaneously or in an individual iterative fashion) to try and improve compactness. For example, in sources [30], [56], [60], the authors start with all objects spread out (without overlap) in the packing space and then use physics inspired approaches to simultaneously draw the objects together. [30] simulates a 'rubber band' stretched around all the objects which then draws them together under elastic forces, [56] uses dynamic vector fields to calculate object accelerations that consider both the packing objective (to draw objects together) and physical interactions between objects, and [60] uses attraction-based accelerations between objects which vary depending on the distance between them.

Another common technique employed in parallel packing approaches involves relaxing intersection constraints between objects in the packing space (to allow some overlap between objects) and then iteratively translating/rotating objects to remove overlap. For example, in [22], the authors start with the objects spread out in a large container and then shrink the container, translating any object outside the new container boundary inside. This process often induces overlap between the packed parts, which the algorithm then tries to iteratively remove (by calculating travel directions/rotations for the objects which lead to a reduction in overlap). If an overlap free solution is found, the container is shrunk further and the process repeats, only stopping when the container cannot be shrunk further without causing irremovable overlap. The idea with such approaches is that by starting with a compact solution where some violation of the overlap constraint is allowed, the algorithm should converge to feasible local optima faster.

A point worth noting is that iterative approaches involving a relaxation of intersection constraints are more prevalent in metaheuristics packing algorithms, such as in [64-66] where authors use simulated annealing to iteratively translate/rotate or swap objects to remove overlap. Without the use of metaheuristic search techniques to help avoid local entrapment, such techniques become very sensitive to the choice of initial locations for the objects. To help avoid this problem, the one constructive approach which uses overlap reduction, [22], restarts the packing algorithm 100 times with different random starting locations.

A more unique example of a parallel packing approach can be found in [54]. In this paper, authors start by partitioning the packing space into cells where cell sizes are proportional to the size of the object assigned to it. The objects are then inserted into the cells at 10% of their original size. The algorithm then tries to simultaneously 'grow' the objects in their

respective cells to their original size, whilst allowing the objects to rotate. They also allow objects to be swapped in the packing space or replaced with unpacked objects, and use a hole filling method to fill any remaining gaps in the structure with unpacked objects.

One of the main benefits of parallel approaches compared to serial approaches (i.e. one-by-one packing approaches) is their flexibility in determining object placement. Unlike serial approaches which are limited by their choice of placement heuristic, objects in parallel approaches can move more freely within the packing space (with the degree of freedom controlled by the allowed rotation/translation step sizes) and are thus able to explore more potential layouts. One drawback of this however, as previously mentioned, is that they can be subject to sensitivity with the choice of initial solution. Furthermore, they also present problems from a real-world perspective in that there is no explicitly defined packing sequence, which can make it difficult to translate the solution into step-by-step instructions for manual or robotic packing. Additionally, incorporating real world considerations, such as object stability and accessibility, is more difficult since these constraints cannot be checked incrementally as each object is added. Without being able to check these constraints incrementally, it is possible, for example, that when trying to recreate the packing structure in the real world, objects might shift in the container halfway through packing, invalidating the solution.

As a final point, it is worth noting that some constructive approaches also employ additional steps to try and find better solutions. For example, in [30], the authors allow temporary volume relaxation of the simulated rubber band stretched around the objects, to allow objects to rotate into more favourable positions. In [59], the authors use a post processing step which involves shaking the container up and down (modelling the motion of objects using rigid body dynamics), with the aim of allowing objects to naturally settle into more compact arrangements.

To summarise, constructive approaches are intuitive and computationally efficient approaches for incrementally building packing solutions from an initially empty configuration, trading global search capability for rapid solution generation. Regarding the real-world application of waste packing, serial approaches (where objects are packed one by one) are arguably more suitable when compared to parallel approaches (where all objects start dispersed in the container and are then iteratively optimised). This is due to their ability to produce clear, reproducible packing orders (which can then be carried out by humans or robots) and their ability to easily incorporate constraint checking as objects are incrementally added. Furthermore, such approaches can easily be coupled with metaheuristic search techniques (e.g. to optimise the packing order) allowing them to

better explore the search space and avoid poor local optima entrapment. In contrast, parallel approaches, while having greater flexibility with the placement of objects, suffer from sensitivity to initial configurations and difficulty with constraint checking. Furthermore, they do not define an explicit packing order, making their solutions more difficult to translate to real-world settings.

2.2.2 Metaheuristic Approaches

Metaheuristic approaches are high-level optimisation strategies designed to explore complex search spaces efficiently, offering a practical means of escaping local optima to find higher quality solutions when compared to their constructive counterparts. Unlike constructive heuristics, which are tailored for specific problem types, metaheuristics are problem independent search strategies, making them applicable across a wide range of problems. Their flexibility and efficient search schemes make them especially useful in 3D irregular object packing problems where the search space is vast and exhaustive search is computationally impractical. Table 2-3 below presents publications that use metaheuristic approaches for 3D irregular packing.

TABLE 2-3. Publications based on metaheuristic approaches.

Source - (year)	Object Representation	Object Complexity	Container Type	Packing Method	Constraints	Rotation Scheme
[64] - 1997	Cuboid/cylinder approximation	Low	Single container (bounded & unbounded)	Objects start randomly dispersed in packing space (overlap allowed). Simulated annealing used to iteratively translate/rotate/swap shapes to improve packing.	No overlap. Containment. Problem specific.	Finite (90°)
[65] - 1997	Cuboid/cylinder approximation	Low	Single container (unbounded)	Objects start randomly dispersed in packing space (overlap allowed). Simulated annealing used to iteratively translate/rotate/swap shapes to improve packing.	No overlap.	Finite (component specific)
[66] - 1998	Octree	High	Single container (bounded)	Objects start randomly dispersed in packing space (overlap allowed). Simulated annealing used to iteratively translate/rotate/swap shapes to improve packing.	No overlap. Containment. Problem specific.	Finite (varied by simulated annealing)
[61] - 2001	Voxel	High	Single container (bounded)	One by one placement with bottom-centre placement heuristic. Genetic algorithm optimises packing order.	No overlap. Containment. Problem specific.	Finite (5° and 30° about x and y axis)

[67] - 2002	Cuboid approximation	Low	Single container (bounded)	Objects start randomly dispersed in packing space (overlap allowed). Simulated annealing used to iteratively translate/rotate shapes to minimise height and reduce overlap.	No overlap. Containment.	Finite (90°)
[32] - 2002	Voxel	High	Single container (bounded)	One by one placement with fixed packing order (largest vol to smallest). Simulated annealing used to optimise position of each object.	No overlap. Containment.	None
[68] - 2004	Cylinder approximation	Low	Single container (bounded)	Initial layout generated. Simulated annealing used to iteratively translate/rotate individual shapes.	No overlap. Containment. Problem specific.	Finite (component specific)
[69] - 2005	Mesh	Med.	Single container (bounded)	One by one placement with 'attachment point' placement heuristic. Distributed genetic algorithm optimises packing order and part orientations.	No overlap. Containment. Problem specific.	Finite (90°)
[70] - 2005	Octree	Low	Single container (bounded)	Objects start randomly dispersed in packing space (overlap allowed). Pattern search algorithm used to rotate/translate objects to reduce overlap and packing structure height.	No overlap. Containment.	Finite (step size controlled by pattern search algorithm).
[71] - 2006	octree	High	Single container (bounded)	Theoretical framework for algorithm presented in [70].	No overlap. Containment.	Finite (step size controlled by pattern search algorithm).
[72] - 2007	Mesh	Low	Single container (unbounded)	Shapes randomly dispersed in container (overlap allowed). Overlap reduction run. If unsuccessful, guided local search used to escape local optima. If successful, container height decreased, objects moved inside, and overlap reduction run again.	No overlap. Containment.	None
[62] - 2008	Voxel	High	Single container (unbounded)	One by one placement with bottom-left-back placement heuristic. Genetic algorithm optimises packing order and part orientations.	No overlap. Containment.	Finite (user-defined set + 45° rotation about z-axis)
[73] - 2008	Cuboid approximation	Low	Single container (unbounded)	One by one placement object height minimisation placement heuristic. Genetic algorithm optimises packing order and part orientations.	No overlap. Containment.	Finite (90°)
[74] - 2009	Mesh	Low	Single container (unbounded)	Same process as [72]. Additional problem specific constraints added.	No overlap. Containment. Problem specific.	None
[75] - 2009	Mesh	Low	Single container (unbounded)	Same process as [72]. [72] was a preliminary study, this fully generalises the algorithm to 3D.	No overlap. Containment.	None
[63] - 2010	Voxel	High	Single container (bounded)	One by one placement with bottom-left-back placement heuristic. Genetic algorithm optimises packing order and part orientations.	No overlap. Containment.	Finite (with limited free rotation about each axis)

[76] 2012	Cluster of parallelepipeds	Low	Multi-container (bounded)	One by one placement with first fit placement heuristic. Hybrid genetic algorithm/tabu search used to optimise packing order and orientation.	No overlap. Containment. Problem specific.	Finite (90°)
[77] - 2014	Voxel	High	Single container (unbounded)	One by one placement with bottom-left-front placement heuristic. Genetic algorithm optimises packing order and part orientations.	No overlap. Containment.	Finite (32 orientations)
[23] - 2014	Mesh	High	Single container (unbounded)	Input model decomposed into parts. One by one placement with height field minimisation placement heuristic. Order optimisation with tabu search. Merging of parts allowed to minimise number of parts.	No overlap. Containment.	Finite (30°)
[52] - 2015	Mesh	Low	Single container (unbounded)	One-by-one placement to minimise centre height of structure. Packing order optimised by simulated annealing	No overlap. Containment.	Finite (45°)
[78] - 2015	Sphere-tree	High	Single container (unbounded)	Start with initial configuration. Objects iteratively translated/rotated to minimise height. Simulated annealing controls translation/rotation step size.	No overlap. Containment.	Finite (step size varied by simulated annealing)
[79] - 2018	Voxel	Med.	Single container (unbounded)	One by one placement with back-bottom-left placement heuristic. Chaos firefly algorithm used to optimise packing order and part orientations.	No overlap. Containment.	Finite (90°)
[80] - 2020	Mesh	Low	Single container (unbounded)	One by one placement with deepest-bottom-left placement heuristic. Genetic algorithm optimises packing order.	No overlap. Containment.	Finite (90°)
[81] - 2021	Mesh (from point cloud)	High	Single container (bounded)	Objects randomly dispersed close to container boundary. Simulated annealing used to minimise bounding volume by translating/rotating all objects simultaneously.	No overlap. Containment.	Finite (0.1°)
[82] - 2020	2D projection	High	Multi-container (bounded)	One by one placement with back-bottom-left placement heuristic. Genetic algorithm optimises packing order and allocation of objects to containers.	No overlap. Containment.	Finite (z-axis rotation only, 90°)
[83] - 2022	Cuboid approximation	Low	Multi-container (bounded)	One by one placement with deepest-bottom-left-fill placement heuristic. Genetic algorithm optimises packing order and orientation of parts.	No overlap. Containment.	Finite (90°)
[21] - 2022	Cuboid approximation	Low	Multi-container (bounded)	Extension of work presented in conference paper [83]. Same methodology used.	No overlap. Containment.	Finite (90°)
[84] - 2023	2D projection	High	Multi-container (bounded)	One by one placement using bottom left fill. Novel placement heuristic to determine object rotation. Packing order optimised by genetic algorithm.	No overlap. Containment.	Finite (number of angles and values set by heuristic)

[85] - 2023	Granulated convex hull of point cloud	Med.	Multi-container (bounded)	One by one placement. Deep Q-network used to optimise the number and type of boxes, packing sequence and object orientations.	No overlap. Containment.	Finite (z-axis rotation only, 90°)
[31] - 2023	Voxel	High	Single container (unbounded)	1st packing method: one by one placement using 'No Fit Voxel' to find potential sites. Packing order and placement heuristic choice optimised by iterated local search.	No overlap. Containment.	None
[31] - 2023	Voxel	High	Single container (unbounded)	2nd packing method: Initial layout generated. Container height shrunk to induce overlap. 'Strategic oscillation' switches between overlap reduction and modifying container height. Tabu search to optimise object positions in overlap reduction step.	No overlap. Containment.	None
[31] - 2023	Voxel	High	Single container (unbounded)	3rd packing method: Same as 2nd packing method but using variable neighbourhood search instead of tabu search.	No overlap. Containment.	None
[28] - 2025	2D projection	Low	Single container (bounded)	Input model partitioned and cut parts projected onto 2D plane. One by one placement using no fit polygon and bottom left fill strategy. Packing order optimised using genetic algorithm.	No overlap. Containment.	None

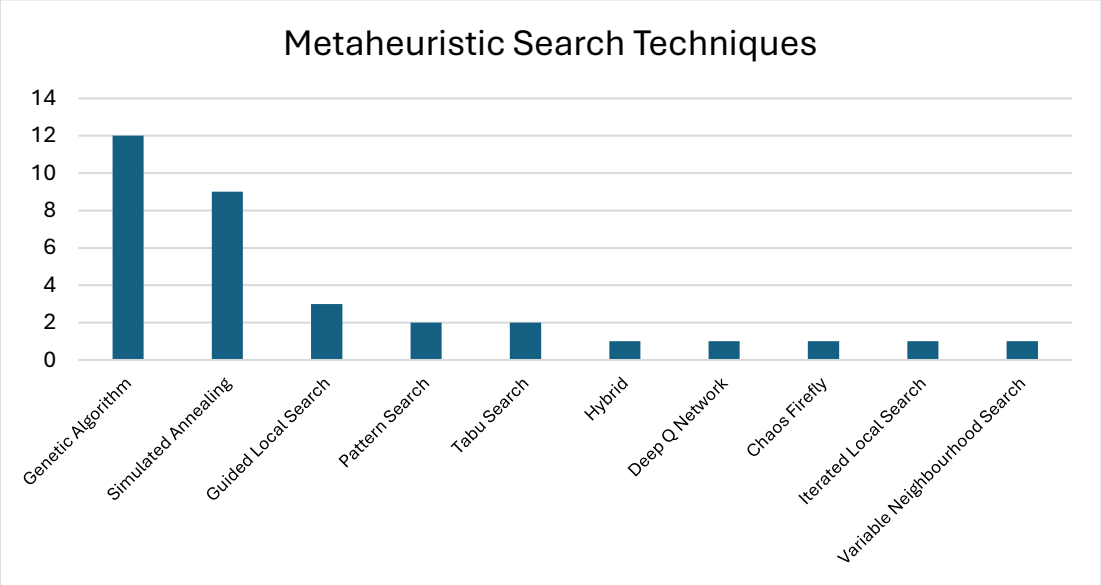


Fig. 2-7. Bar chart showing number of occurrences for each metaheuristic in the literature presented in Table 2-3.

The bar chart presented in Fig. 2-7 shows the number of occurrences for each type of metaheuristic search technique used by the algorithms presented in Table 2-3, ordered in descending number of occurrences. From this figure it is evident that the two most used

metaheuristics are Genetic Algorithms (GA) and Simulated Annealing (SA), with the occurrences for these two metaheuristics alone being more than the occurrences of all the other metaheuristics combined.

Genetic algorithms are population-based optimisation methods inspired by the principles of natural selection in Darwinian evolution [86]. They work by maintaining a pool (or 'population') of candidate solutions to the problem, which are 'evolved' over successive generations. In each generation, solutions are selected based on their fitness (i.e. how 'good' a solution is) after which, new solutions (or 'child' solutions) are generated via crossover and mutation. During crossover, two 'parent' solutions are selected, and parts of each parent are recombined to form two child solutions, with the process aiming to mimic biological reproduction and the inheritance of 'good genes' from the reproduction of fitter parents. During mutation, a child solution produced via crossover will be selected (usually with a small probability of selection), and parts of the solution will be randomly altered, mimicking the process of biological mutation within a population. Regarding the packing algorithms listed in Table 2-3 above, the most common application of GAs is for the optimisation of placement order in one-by-one placement methods (e.g. [28], [61], [84]). Some packing approaches also augment the GA by including the orientation of the parts as another optimisation variable, rather than allowing the placement heuristic to decide this automatically (e.g. [62], [63], [69]).

One of the key advantages of GAs is their ability to explore diverse regions of the search space simultaneously due to their population-based nature. This diversity helps prevent premature convergence to poor local optima and makes GAs particularly well suited for complex problems with large search spaces where multiple suboptimal solutions may exist [86]. However, this better ability to explore the search space comes at the cost of higher computational demand (as the objective function must be evaluated for every population member). Furthermore, the algorithm's success can be sensitive to choices inherent in GA design such as the population size, mutation/crossover rate and the mutation/crossover strategy, which often require manual tuning for effective performance.

In contrast to the genetic algorithm, simulated annealing is a *single-solution*, trajectory-based metaheuristic inspired by the physical process of annealing in metallurgy, where materials are heated and cooled to alter their physical properties (e.g. to remove defects) [87]. In the context of optimisation, SA begins with a single solution to the problem and iteratively explores neighbouring solutions by making conservative modifications to the current solution. If the new solution is better, it is accepted and replaces the current solution. If it is worse, it may still be accepted with a probability that decreases over time

(which is controlled by the 'cooling schedule'). This probabilistic acceptance of worse solutions helps the algorithm escape local optima and explore a broader portion of the search space. Regarding the algorithms listed in Table 2-3, the most common application of SA is for the optimisation of object positions in parallel constructive approaches, such as in [64-68], [81] where objects are iteratively translated/rotated (and in some cases allowed to swap), with the acceptance of translation/rotation/swapping moves controlled by the SA algorithm. A slightly different implementation of SA worth noting is in [78], where rather than translating/rotating objects by a fixed step size (with the chance of move acceptance controlled by SA), the authors instead use SA to directly control the step size that objects can take (with the step size decrease controlled by the cooling parameter).

One of the primary benefits of simulated annealing is its operational simplicity and ease of implementation [87]. Furthermore, since it only works on a single solution (rather than a population of solutions), the algorithm will be much faster (when compared to a GA) as it requires fewer evaluations of the objective function. This feature, however, is somewhat of a double-edged sword, since the use of only a single solution also lowers the diversity of the algorithm, which can cause it to become more easily entrapped in poor local optima if the algorithm is configured poorly. As with GAs, SA algorithms are also sensitive to parameter choices (particularly with the choice of cooling schedule and step sizes), which must also be manually tuned for effective performance.

Other less commonly used approaches (as shown in Fig. 2-7) include Guided Local Search (GLS), Pattern Search (PS), Tabu Search (TS), Iterated Local Search (ILS), Variable Neighbourhood Search (VNS), swarm-based approaches (chaos firefly), hybrid algorithms and Artificial Intelligence (AI).

GLS, ILS, TS and VNS are all modified versions of simple local search, which starts with a candidate solution and then iteratively moves to neighbouring solutions (with a neighbourhood defined as a set of potential solutions which are minimally different to the current candidate solution) [87]. The problem is that local search is a greedy process, only accepting moves which lead to an improvement on the current solution. To help alleviate this greedy behaviour several modifications have been proposed:

ILS is the simplest modification of local search and, as the name implies, functions by calling the local search process in an iterative fashion, using a different starting location each time [87]. For example, in [31], one of the algorithms presented is a one-by-one placement approach where local search is performed by modifying the placement heuristic of a single object and then re-packing the parts, with the starting solution reinitialised using

a ‘kick’ upon local optima convergence. The kick involves performing several position swaps in the packing order sequence (as well as changing the placement rules associated with the objects) at random and then re-running the local search process.

VNS functions similarly to ILS in that it repeatedly restarts the search, but with the key difference being that it uses multiple pre-defined neighbourhood structures during the local search [87]. As stated, a neighbourhood refers to a set of solutions which can be reached from the current one by making a small change (e.g. changing the placement heuristic for a single object). While ILS perturbs the current solution to escape local optima, it always stays within a single neighbourhood (e.g. only changing the placement heuristic for one object). In contrast VNS systematically changes the neighbourhood itself to allow the algorithm to explore wider areas of the search space. For example, in [31], one of the presented algorithms is a parallel VNS-based packing approach (where an initial layout is iteratively optimised by translating objects) which contains several neighbourhood structures, two of which are ‘enclosing cube neighbourhood’ and ‘axis aligned neighbourhood’. The enclosing cube neighbourhood defines a local search area as a cube of voxels around an items position, allowing movement in all directions within a small, specified range (e.g. 1 voxel). In contrast, the axis aligned neighbourhood restricts movement along one of the principal axes (x, y or z) but typically allows for larger travel distances. Both approaches offer different ways to explore the search space, one by making small adjustments to the objects position in 3D space (enclosing cube neighbourhood) and the other by making larger axis-aligned translations of the object to a new position in the packing space (axis aligned neighbourhood).

TS is more advanced than VNS and ILS, incorporating a memory-based mechanism to prevent solution cycling and encourage better exploration of the search space. The memory function is implemented as a ‘tabu list’ to store a set of the most recently visited solutions, which are temporarily prohibited to avoid the algorithm retracing its steps to soon [87]. For example, in [23], authors use a one-by-one placement approach with tabu search to optimise the packing order. The tabu list stores previously visited packing orders which have led to an improvement in the objective function, preventing the algorithm from revisiting these solutions for a certain number of (user defined) iterations.

GLS is the most advanced of the four approaches, introducing dynamic penalty functions to penalise certain features or characteristics of the current solution which are believed to contribute to its poor quality [87]. These penalties are incorporated into the objective function, reshaping the optimisation landscape to help encourage the algorithm to explore less visited areas of the search space. For example, in [75], authors present a GLS-based

packing algorithm which optimises an initial layout by iteratively translating objects in axis aligned directions to reduce overlap. When the search encounters an infeasible local optimum (i.e. when no further axis-aligned moves can reduce overlap), a penalty term is added to the objective function. Penalty values are calculated for pairs of overlapping objects that contribute most to the total overlap in the packing structure. Once the penalty term is added, the search continues from the same layout with the new objective function. Now, with the augmented penalty term, the algorithm is motivated to separate the most problematic overlaps, even if doing so induces temporary overlap elsewhere.

Despite their proven use in solving 3D packing problems, the four approaches (ILS, VNS, TS and GLS) all suffer from several common drawbacks. First, given that these algorithms work on a single solution at a time (rather than maintaining a population of solutions), they can be more susceptible to initialisation, especially in large or complex problem spaces (e.g. packing problems with many items). Second, these methods are primarily local search-based, which limits their exploration ability, making them susceptible to becoming trapped in local optima despite mechanisms like variable neighbourhoods, restarts and memory. Finally, all four approaches have parameters that require careful tuning, such as penalty weights, neighbourhood sizes or memory lengths, with the performance of the algorithms depending heavily on these parameters.

Pattern search is a direct search method that explores the search space by evaluating a set of pre-defined movement patterns, with a ‘pattern’ representing a distinct way to modify the current solution [88]. The algorithm tests all the patterns, identifying the most promising search direction based on the effect each movement pattern has on the objective function. Unlike local search methods which explore neighbouring solutions one small move at a time, accepting the first (or best) move which leads to an improvement, pattern search simultaneously evaluates a set of different moves before selecting the most promising one. For example, in [70], [71], authors present a parallel packing algorithm (where all objects start dispersed in the container with overlap allowed), where pattern search is used to iteratively translate/rotate objects to reduce overlap. Instead of directly testing all patterns and then selecting the best one, they use a novel function to estimate which pattern will likely lead to the most reduction in overlap (factoring in object translation/rotation as well as the step size of these moves). This allows them to speed up the algorithm, since estimation of the cost function for different moves is computationally cheaper than calculating the cost by testing all the moves. The main downsides with PS, as with the four local search algorithms, is that it has limited global search ability due partly to its greedy and deterministic nature (only accepting solutions which improve the objective function

value), and because it only works on a single solution at a time (meaning it lacks the diversity benefits of population-based approaches).

In contrast to the previous single-solution algorithms, swarm based algorithms (such as the chaos firefly algorithm used in [79]) are population-based algorithms inspired by the collective behaviour of natural systems such as bird flocks, fish schools or insect colonies [87]. These algorithms solve optimisation problems by having a group of simple agents (e.g. fireflies) explore the solution space in a cooperative fashion, with the movement of the agents guided by their own best-known position in the search space as well as the entire populations best-known positions. This allows the agents to adapt their search direction in the search space based on local and global knowledge of the landscape. For example, in [79], authors propose a one-by-one packing algorithm where a firefly algorithm is used to optimise both the packing order and orientation of parts. Each firefly encodes a packing order and a list of orientations for each part. The algorithm evaluates each packing solution and then adjusts the 'brightness' of each firefly in proportion to its fitness (with fitter individuals being 'brighter'). During the search process, fireflies are attracted to other fireflies based on their brightness and distance between them (with attraction degrading with distance). To handle the discrete nature of the problem, the algorithm uses the Smallest Position Value (SPV) rule [89] to map the continuous position vector of each firefly (defined in an n-dimensional search space where n is the number of parts) to a discrete packing sequence. Additionally, chaotic maps are integrated into the algorithm to enhance exploration by introducing controlled randomness in the movement updates, helping the search escape local optima.

Of the two population-based metaheuristics discussed here (the other being GA), there is a distinct lack of packing papers utilising swarm-based approaches, with most papers choosing GA instead. There are likely several reasons for this. Firstly, with swarm-based algorithms, since the position of each agent is encoded using continuous variables, an additional step is required to map the continuous variables to discrete space (e.g. a packing sequence). In contrast, the chromosome-based encoding scheme used by GAs can easily represent discrete sequences such as packing orders or part orientations. Secondly, GAs benefit from having a crossover operator, which enables the inheritance of advantageous traits (e.g. a good partial placement sequence) from the parent solutions. In contrast, swarm-based approaches are limited to gradual position updates based on attraction dynamics, which limits their ability to exploit promising solutions. Finally, GAs present a more modular solution approach, with components such as the selection operator for parent selection, crossover operators and mutation strategies easily being swappable with

other ones to suit the problem at hand. This factor, coupled with the vast array of literature on GAs makes them very versatile and flexible tools that can be applied to most optimisation problems.

A final point worth noting is the distinct lack of hybrid and AI approaches in existing literature. With hybrid approaches, the aim is to combine multiple metaheuristic approaches to try and leverage the benefits of each. For example, in [76], authors propose a hybrid GA/TS algorithm for optimising packing order and part orientations, which leverages the search benefits of a GA, combined with TS to prevent solution cycling to promote better exploration of the search space. In contrast to traditional metaheuristic search techniques, AI approaches aim to leverage capabilities such as self-learning and adaptive decision making, to solve complex optimisation problems. For example, in [85], authors propose a multi-container packing algorithm where reinforcement learning is used to optimise the packing sequence, part orientations and choice of container. A key advantage of reinforcement learning in this context is that it does not require any prior knowledge of the problem (i.e. it doesn't require training with pre-labelled datasets). Instead, it learns how to construct effective packing solutions by interacting with the environment through a predefined set of actions, guided by a rewards scheme which evaluates to quality of solutions and set of actions taken to achieve it, thereby encouraging beneficial decision making as the search progresses.

To summarise, it is clear from this review that when it comes to metaheuristic optimisation of packing, there are many tried and tested options available to the user. The primary downside however is that, given the limited amount of consistent benchmarking within packing literature (with many authors choosing to use their own object datasets), it is difficult to say with certainty if one metaheuristic approach is better than another. The two approaches which stand out, however, as the most used are genetic algorithms and simulated annealing, with the frequency of their usage dominating other approaches used in literature. Genetic algorithms offer great flexibility with their implementation, allowing them to easily be adapted to a range of problems whilst also offering good global search ability due to their population-based nature. Simulated annealing acts as a single-solution counterpart to GAs, being much faster and simpler to implement, but with the downside of reduced global search ability due to not using a population of solutions. Additionally, both approaches are very well studied in literature [90], [91], meaning they are well established and understood technologies, with the added benefit of having much literature to draw on in terms of modifications for improvement. As a final point, a distinctive lack of hybrid optimisation approaches (where metaheuristics are combined to enhance their search

ability) and artificial intelligence approaches (where AI is used to independently 'learn' how to solve the problem) was also noted. As such, both areas present good opportunities for future research (in particular, trying to hybridise metaheuristics with AI).

2.2.3 Discussion

Having presented and discussed relevant examples of heuristic and metaheuristic packing algorithms in 3D irregular-object packing literature, the following section presents a discussion on the algorithm design choices most relevant to real-world waste packing.

The discussion is organised around five key themes that strongly influence both computational effort, solution quality and practical fidelity: 1.) **Object representation** and its impact on computational cost and geometric fidelity, 2.) **Object complexity** and how this affects choice of object representation, 3.) **Container type** (single vs multi-container, bounded vs unbounded) and implications for deployability, 4.) **Constraints** relevant to real-world packing and how they are enforced, and 5.) **Rotation** handling (from discrete angle sets to continuous sampling) and its impact on computational cost/solution quality.

2.2.3.1 Representation of objects

When it comes to packing optimisation, one of the key factors to consider (particularly from a computational complexity perspective) is how the objects are represented within the algorithm. Unlike simple shapes which can easily be modelled mathematically (e.g. spheres, cylinders, cones, etc.), one of the key challenges in 3D irregular object packing is how to model the complex surface geometry of highly irregular objects.

Over the years, researchers have proposed multiple approaches to modelling 3D objects, which may be broadly classified into three different categories:

- Simplified representation – e.g. 2D projection, bounding volume approximation (bounding box, cylinder, sphere etc.).
- Volumetric representation – e.g. voxel, octree, parallelepiped approximation.
- Surface representation – e.g. mesh, point cloud.

Fig. 2-8 shows the most frequently used types of object representation (by category) for all the algorithms presented in Tables 2-2 and 2-3.

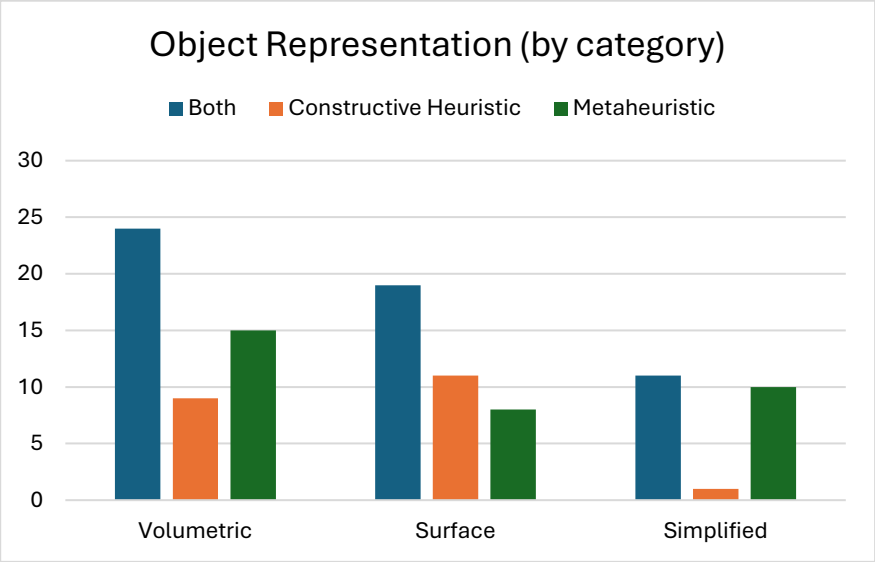
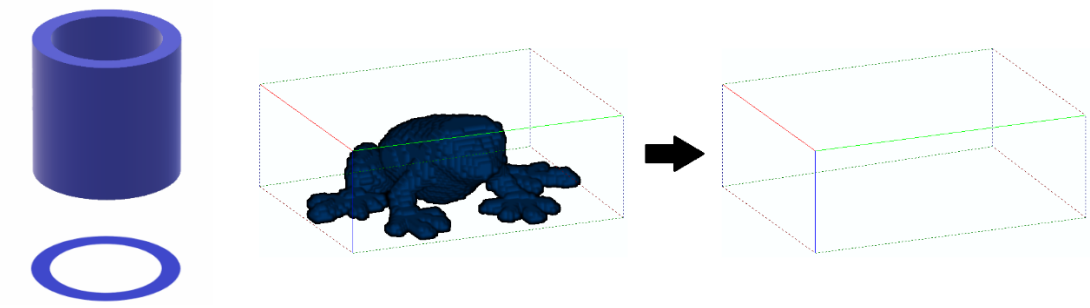


Fig. 2-8. Bar chart showing number of occurrences for each type of object representation in the literature (by category) presented in Tables 2-2 and 2-3.

Simplified Representation



(a) (b)

Fig. 2-9. Examples of simplified representation. (a) – 2D projection of a 3D tube segment onto a flat surface. (b) – bounding box approximation of frog model (model adapted from [92]).

Simplified representations are the most straightforward and computationally efficient methods for representing objects. The most common simplified representations are 2D projection (where a 3D shape is projected onto a 2D surface, transforming the 3D packing problem into a simpler 2D one), and bounding volume approximation (where objects are represented by a simple shape, such as a cuboid, sphere or cylinder, which bounds its volume). Fig. 2-9 shows examples of 2D projection and bounding approximation.

The primary benefit of simplified representations is the increase in computational efficiency due to a reduction in the complexity of overlap checking between parts. The checking of overlap between packed objects in a container is by far the most frequent operation

performed in packing optimisation, accounting for approximately 90% of computational complexity [63]. Using simplified representations allows for much faster overlap checking between objects by reducing the geometric complexity involved in these checks.

For example, with 2D projections, algorithms can make use of the popular No Fit Polygon (NFP) approach (commonly used in 2D packing) for determining feasible placement positions for objects without overlap. The NFP represents all relative positions where one objects is in contact with another without intersecting, effectively encoding all valid placements around a fixed object [45]. Its primary strength lies in its ability to determine placement locations for objects without having to perform repeated geometric intersection testing. As such, its use is very prominent in 2D packing algorithms, with multiple fast and robust approaches existing for its computation [93-95]. It is worth noting that, whilst use of the NFP method in 2D is an efficient and well-established technology, its application to 3D packing remains very limited, with only 2 studies existing where this has been attempted [31], [53]. As such, the adaptation of the NFP to 3D (in particular, how to calculate the NFP in a fast and efficient manner), remains an open challenge in 3D packing research.

Similarly with bounding volume approximations, due to the use of low complexity shapes which can easily be modelled mathematically, the task of intersection checking is reduced to simple arithmetic comparisons. This avoids the need for mesh-level (or voxel level) collision detections which involve more computationally intensive operations such as triangle-triangle intersection testing (with triangular meshes) or voxel grid occupancy checks (with voxels), which grow more computationally expensive with an increase in the complexity of objects.

Whilst simplified representations can greatly reduce computational complexity in packing problems, they are not well suited for arbitrary packing problems involving complex geometries. The problem with these representations is that they discard critical volumetric and geometric detail about the objects, which is essential to consider for efficient packing. For example, with 2D projections, by reducing the complex 3D geometry of an object to a simplified 2D footprint on a plane, critical information about the object, such as its height and internal cavities, are ignored. Similarly with bounding approximations, by abstracting objects to simple 3D shapes (e.g. cuboids, spheres, cylinders), critical information about the objects surface geometry is also ignored.

A key issue with these simplified representations is the loss of nesting potential, i.e. the ability to fit shapes into one another (or ‘nest’ them) based on concave/convex regions on the objects surface geometry. By simplifying the shapes, key information needed to perform

nesting (i.e. information about the objects surface geometry) is lost. As a result, performing packing with simplified representations (particularly when the original objects are highly complex and irregular) will likely lead to inefficient packing solutions with large gaps between objects [61].

Another issue with simplified representations is the difficulty with implementing real-world constraints such as object stability (ensuring objects remain stable within the packing structure) and robotic packability (whether objects can be grasped by a robot and placed in the desired location). Such constraints often rely on precise information about the geometric surface of the object, for example, to calculate contact forces between touching objects or to determine grasping locations on an objects surface for a robotic manipulator.

As a final point, it can be seen from Fig. 2-8 that the use of simplified representations is far more prevalent in metaheuristic approaches compared to constructive approaches. Given that metaheuristic packing algorithms often involve iteratively evaluating multiple candidate solutions (e.g. multiple packing orders in order optimisation), it naturally follows that researchers would seek ways to represent objects in more computationally efficient manners.

Volumetric Representation

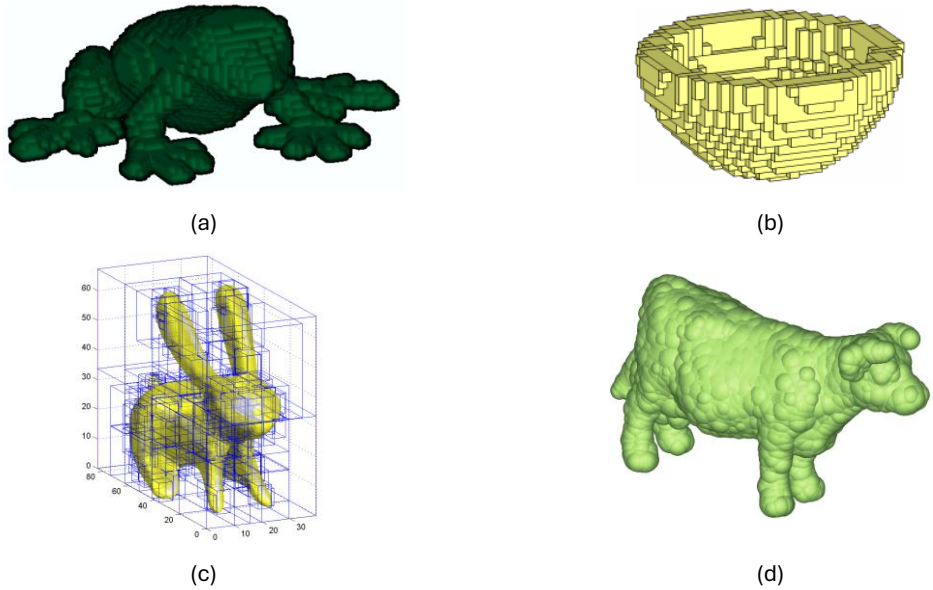


Fig. 2-10. Examples of volumetric representation. (a) – Voxelised model of frog (adapted from [92]). (b) – Parallelepiped approximation of bowl (from [57]). (c) – Octree decomposition of rabbit (from [77]). (d) – Sphere tree decomposition of cow (from [78]).

Volumetric representation involves the discretisation of objects into a finite collection of discrete (and simple) volumetric shapes (e.g. cubes, spheres, parallelepipeds) which approximate the input object. Fig. 2-10 shows examples of different volumetric approximations that have been used in 3D packing.

The most common type of volumetric representation is the voxel, where an input model is discretised into uniform cubic units on a grid. Since each voxel represents a uniform volume of space, this allows objects to be stored as 3 dimensional Boolean arrays, with 1 representing an occupied grid cell, and 0 representing vacant space. The primary benefit of this is that such data structures are straightforward to implement and efficient work with [96]. For example, checking for collision between voxelised objects can be reduced to simple bitwise operations on the binary matrices, where collision occurs if the same voxel (i.e. grid cell) is marked as occupied by both objects [63].

The main downside however with voxel approaches is that the memory requirement for representing objects can increase exponentially with the size of the objects. For example, consider a simple solid cube of 10x10x10 voxels, with a volume of 1000. If the dimensions of each length are doubled (to 20x20x20), the new volume of the cube would be 8000 (resulting in an eightfold increase in memory usage).

Another approach closely related to voxels is octree decomposition. Octree decomposition works by recursively decomposing the 3D space into a hierarchical tree data structure, with each node split into eight child nodes, each representing a smaller subregion of the original volume [77]. In doing so, the model is split into cubic units (much like with voxels), but with the main difference being the ability to vary the resolution of the decomposition at different parts of the model. The benefit of this is that it allows for fine grained detail in complex areas (such as highly curved surfaces that are poorly approximated by course voxels), while maintaining lower resolution in simpler areas (such as flat or empty spaces), thereby improving memory efficiency. An interesting point to note is the work in [78], where rather than cubes, the authors propose a novel decomposition algorithm to decompose the structure into spheres (which they refer to as sphere tree decomposition).

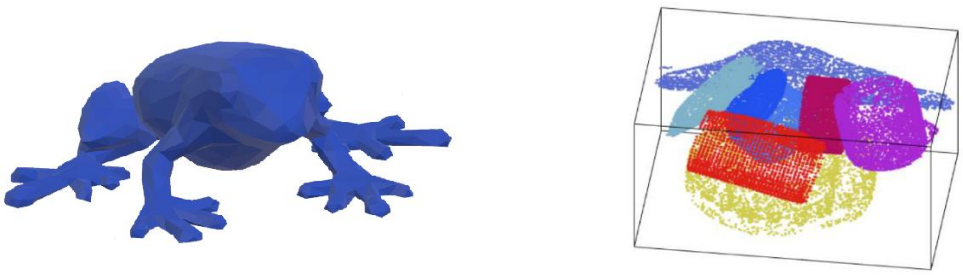
Whilst having the potential to be more memory efficient than fixed size voxel approaches, octree decomposition still presents several challenges. Firstly, whilst the memory usage may be lower, tree data structures are more complex to construct and traverse (when compared with simple binary arrays) which can lead to longer data access times [96]. Additionally, if the decomposition resolution is not uniform (i.e. to adapt the level of detail based on surface complexity), additional computation is required to decide which cells

should be further subdivided and which can remain course, increasing preprocessing overhead [97].

Another less used approach is parallelepiped decomposition, which is similar in appearance to octree decomposition, but with the key difference that the volumetric units are not constrained to cuboids but can be any parallelepiped (e.g. rectangles). The principle of parallelepiped representation is similar to multi-level octree decomposition in that it can allow for different levels of detail and can reduce memory usage, but the issue is that, unlike octree decomposition, there is no general method for decomposition, with methods often being proposed by the authors themselves (e.g. [57]).

From Fig. 2-8, much like with simple representations, an increased use in volumetric representation for metaheuristic approaches is noted. As with simplified representations, this is likely due to the increased computation associated with using metaheuristic approaches and the fact that volumetric approaches offer a good trade-off between computational speed and level of object detail. Whilst simplified representations are the best in terms of speed, they suffer from very poor approximation of the object's geometry. In contrast, volumetric-based models preserve a greater degree of surface detail (compared to simplified representations), whilst still allowing for computationally efficient operations like overlap checking.

Surface Representation



(a) (b)

Fig. 2-11. Examples of surface representation. (a) – Triangular mesh model of a frog (obtained from [92]). (b) – Packed objects represented using point clouds (image from [20]).

As opposed to volumetric representation, where objects are modelled as a collection of discrete volumetric units (e.g. cubes, spheres, parallelepipeds), surface representation captures the geometry of objects by instead defining them as a surface boundary. The most common of these is triangular mesh representation (e.g. Fig. 2-11 a), where the objects surface is modelled as a collection of triangular facets. The primary benefit of this representation is that it represents an industry standard, with many 3D modelling software

(such as computer aided design packages) using this representation. As such, packing algorithms developed using mesh representation often require no form of pre-processing of the shapes (e.g. to convert them to voxel representation). The main downside however is that intersection checking is typically more computationally expensive (especially for highly detailed models with many facets), since it requires pairwise checking of planar faces of two objects to check if any of the triangular faces intersect.

A much less common form of surface representation is the point cloud (e.g. Fig. 2-11 b). Point clouds are a collection of points on an objects surface obtained from 3D scanning processes. The main benefit of point clouds are that they allow real world objects to be scanned and converted into digital counterparts. The main problem with point clouds however is that they are difficult to work with directly since intersection checking becomes very difficult when dealing with point coordinates with zero volume [85]. As such, all the papers presented in this study (which use point clouds), first convert the point clouds into some form of surface or volumetric representation before packing them (e.g. by modelling the individual points in the cloud as spheres [20], or by converting them to a convex hull representation [85]). Due to the real-world applicability of point clouds and their lack of use in 3D packing literature, the use of point clouds in packing remains an interesting avenue for future work.

2.2.3.2 Object Complexity

Closely related to the point of object representation is object complexity. Object complexity plays a crucial role in both the performance and applicability of 3D packing algorithms. In many real-world scenarios (such as waste management, logistics or manufacturing), the objects being packed can often be highly irregular shapes with non-uniform sizes and high variation in surface features. As discussed in the prior section, these characteristics can pose significant challenges for packing optimisation as they increase the difficulty in accurately modelling object geometry, checking for object intersections in the packing space, and exploiting spatial arrangements like nesting. As such, understanding how different packing algorithms handle geometric diversity is essential for evaluating their robustness and suitability in practical applications.

Complexity refers to the surface geometry of shapes, considering factors like convexity, symmetry and surface variation in curvature. For classifying geometric complexity, the classification system proposed in [98] is adopted, which classifies objects based on the

aforementioned factors (convexity, symmetry and surface variation). Their classification is based on two key statements:

- *"A convex object with no variation in surface curvature and multiple axes of symmetry is considered simple"* [98].
- *"A concave object with variation in surface curvature and little to no symmetry is considered complex"* [98].

Using these observations, they proposed the following three conditions for classifying object complexity:

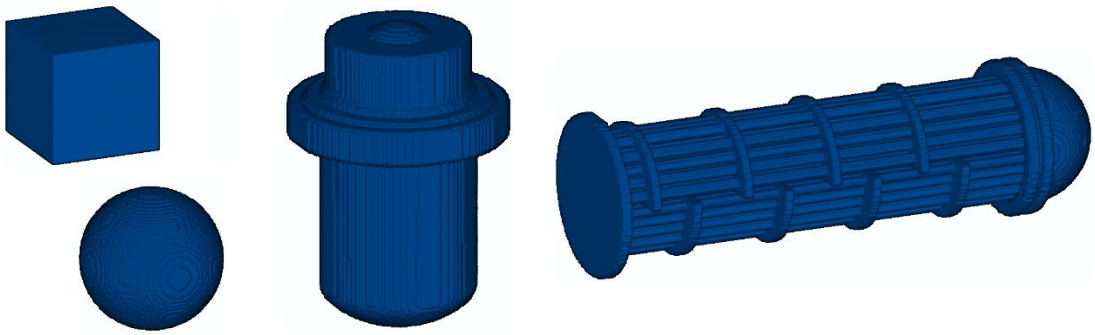
1. The object is concave.
2. The object has few, or no axes of symmetry.
3. The object has variation in the curvature of its surface.

The level of an object's complexity can thus be described based on how many of the conditions are met. If a shape meets only one (or none) of the criteria, it is considered simple. If it meets two of the three criteria, it is classed as intermediate. If it meets all three criteria, it is classed as high.

For example, Fig. 2-12 (a) depicts a cube and a sphere. Whilst the cube meets condition two (few axes of symmetry), it does not satisfy conditions one and three (the cube is convex and has no variation in surface curvature). The sphere meets condition three (variation in surface curvature) but does not meet conditions one and two (it is convex and has infinite lines of symmetry about the centroid). As such, both are classified as simple.

Fig. 2-12 (b) shows a simple reactor pressure vessel. The model has infinite symmetry about the central z axis (meaning it does not satisfy condition two) but is also concave and has variation in the surface curvature (conditions one and three). As such, it is classed as intermediate.

Fig. 2-12 (c) shows a heat exchanger pipe bundle. This model is concave, has no axes of symmetry and has variation in the surface curvature. As such, it satisfies all three conditions and is considered highly complex.



(a) (b) (c)

Fig. 2-12. Examples of different complexity levels. (a) – Simple shapes. (b) – Intermediate shape (model from [99]). (c) – Complex shape (model adapted from [100]).

Regarding object complexity, the main point to consider from a practical perspective is the trade-off between object fidelity and computational complexity. For example, complex real-world objects often require fine-resolution representations (e.g. dense voxel grids or high resolution meshes) to accurately capture their complex surface geometry. However, this comes at the expense of increased memory usage (to store the object models), and increased computation time (since more overlap checks must be performed when packing). In contrast, using simplified representations for objects (like bounding boxes or 2D projections), can reduce memory requirements and computational complexity, but often at the expense of packing quality (since the algorithm is unable to exploit geometric information to nest objects together).

As such it is essential to consider object complexity when designing a packing algorithm. For example, in [28], the authors use a 2D projection-based approach to pack cut segments of a rector pressure vessel. This simplification is effective in their context, as the cut segments are relatively simple and uniform in size, meaning they can simplify the shapes to speed up the packing without a great loss in fidelity and quality of solution. However, this approach would likely struggle to efficiently pack more complex shapes, limiting its generalisability.

2.2.3.3 Container Type

The characteristics of the packing space play a crucial role in the formulation and complexity of 3D irregular object packing problems. One key consideration is whether the container is bounded (has fixed dimensions) or unbounded (where one or more of the dimensions may expand to accommodate the objects). Another key consideration is whether objects require packing into a single container only, or distribution across multiple

containers. Both factors will directly impact how solutions are constructed and evaluated, as well as affecting the type of constraints the algorithms must deal with.

As stated, the term ‘bounded’ refers to objects being packed into a finite space (i.e. a space with fixed dimensions/volume). Such cases often arise in real world packing applications, such as additive manufacturing (e.g. [22], [24], [84]), transportation of goods (e.g. [49], [51]) and nuclear waste packing (e.g. [21], [28], [83]). The problem with bounded approaches is that, given the finite nature of the packing space, naturally, it is not always possible to pack all the objects. This issue is typically handled in one of two ways in literature; either they assume all objects can fit into the packing space (e.g. [64], [68], [81]), or they only pack a subset of items into the container (e.g. [32]).

Regarding subset selection this can either be done prior or incidentally to the packing. With prior selection, a higher-level algorithm is used to select a subset of objects from the main object pool, which is then packed by the packing algorithm. Naturally, with such approaches the choice of subset will affect how well the parts can pack together; selecting poor subsets may lead to poor space utilisation (or in the case of multi-container packing, more containers). In contrast, with incidental approaches, the selection of objects occurs implicitly during packing, typically through one-by-one packing where objects are packed sequentially until the container is full. In such cases, the packing order of objects will determine how many objects can fit into the container and how well they pack together. As a result, the subset of packed objects emerges naturally by truncating the ordered list to only include objects which have been successfully packed.

It is worth noting that of all the bounded container algorithms presented in Tables 2-2 and 2-3, there are no examples where prior subset selection has been used. Instead, all the listed strategies (involving subset packing) use order optimisation with the allocation incident to the packing order. For order optimisation-based approaches, this is a logical choice since the packing order (and hence the allocation) is already being optimised, meaning it is simpler to stop packing once the container is full, rather than using a separate subset selection process. Using a subset selection process with order optimisation would only add to the complexity, requiring a higher-level algorithm to determine object grouping before calling the packing algorithm.

For parallel approaches, where objects start dispersed in the container, they have a notable tendency to assume that all objects will fit into the packing space (e.g. [64], [68], [81]). Whilst such approaches may be suitable for certain applications (e.g. component layout where the placement space is large enough to accommodate all components), this

assumption would naturally be unsuitable for waste packing applications. For example, in [81], the authors present a parallel packing algorithm for waste applications where they assume all objects will fit into the container. Whilst the volume of the subset is smaller than the volume of the packing space (meaning the objects will all theoretically fit), through simulations they demonstrate that, in some cases, their algorithm is unable to find feasible solutions (where all objects are entirely within the container boundary). With such approaches, since there is no explicitly defined packing order, the only solution to this problem would be to couple the algorithm to a subset selection process. To the best of the authors knowledge, there are no 3D irregular-object packing approaches which also utilise subset selection, presenting an avenue for potential future research.

In contrast, unbounded approaches do not use fixed containers, instead using either a container where one or more of the dimensions can expand (e.g. in [22], [31], [79]), to ensure all objects can be accommodated without overlap), or not using a container at all (e.g. [30], [56], [60]) where the goal is purely to maximise the compaction of objects by drawing them together from initially dispersed starting locations). While such flexibility simplifies the problem by ensuring all objects can fit into the packing space, it limits the practical application of such approaches to real-world waste packing problems, where containers have fixed dimensions.

In contrast to single-container packing, multi container packing is essentially an extension of bounded-container subset packing, where the goal is to find a partitioning of the object set into subsets (each packed into a different container), such that the number of containers is minimised.

One of the benefits of one-by-one placement strategies is that they offer a natural way to extend the methodology to multi-container packing. For example, in [21], [83], authors present a one-by-one multi-container packing algorithm for packing segments of a nuclear reactor pressure vessel. They use a common strategy referred to as ‘First-Fit Decreasing’ to pack objects, which works by placing objects into the first container that will accommodate it. If none of the existing containers can accommodate the part, a new container is opened, and the part is added to it.

One of the drawbacks of parallel approaches, as previously discussed, is that they do not place objects sequentially into the container, meaning the extension to multi-container packing is more complex. As with single container subset packing, if parallel approaches are to be used for multi container packing, the algorithm must be coupled with a subset selection process. The downside of this however is that the starting locations of objects in

the packing space and the choice of subset will strongly influence whether the parts can fit into the container. This could potentially lead to packing algorithms having to trial many different subsets with many different starting locations just to feasibly pack a single container.

As with bounded single container packing, there is a notable lack of literature on subset selection for 3D multi-container packing algorithms.

2.2.3.4 Constraints

In packing optimisation, the two most common constraints inherent in the majority of algorithms are:

1. No overlap – Objects in the packing space must not overlap (i.e. intersect) with other objects in the packing space.
2. Containment – In the event that a container is used, all objects must be entirely contained inside it, with no object protruding from the container boundary.

In addition to these fundamental constraints, many packing problems also have additional problem specific constraints (e.g. limits on the rotational inertia of objects packed around a rotating axis [60], or maximum heat limits in the layout optimisation of heat generating components in electronic design [64]). Given that these constraints relate to problem specific requirements which are not relevant to the packing of nuclear waste, these constraints are classified as 'problem specific' in Tables 2-2 and 2-3. A more detailed review on such problem specific constraints in packing can be found in [101]. For the remainder of this section, the focus is on constraints relating to the packing of nuclear waste.

When it comes to the packing of nuclear waste, the three main constraints to consider are:

1.) Stability of the packed pile – will objects remain stable in their desired placement locations.
2.) Manipulation feasibility – can objects be placed in their desired locations without collision with previously packed objects or the container.
3.) Capacity constraints on the containers – regulatory constraints such as radiation and weight limits.

Stability

Stability is a critical constraint for real world packing applications (like nuclear waste packing) where packing structures must not only be space efficient, but also physically viable. Unlike theoretical (i.e. virtual) packing structures, real world implementations of packing solutions must ensure that objects remain stable in the container under gravitational forces. Without incorporating this consideration into the virtual packing process, problems can arise if the packing structure is to be recreated in real life. For example, ignoring stability could result in packing solutions with poorly balanced or unsupported objects. In such cases, if the pile was to be recreated in the real world (e.g. by packing the parts using robotic manipulators), it is possible that such objects may topple or shift from their desired location when they are placed into the container (preventing the packing structure from being reproducible).

To date, the majority of 3D irregular object packing approaches proposed in literature do not consider the stability of objects. For the limited cases where they do, the problem is that it is modelled in an overly simplistic way (e.g. in [49], where the goal of stability is to ensure that each item is supported about its centroid). For certain type of objects, such as simple cuboids, or shapes which can stack together well (e.g. items of furniture as in [49]), such simplistic stability modelling may be acceptable. For the packing of arbitrary (and often complex) geometries encountered in real-world decommissioning however, such simplifications would be unsuitable as they fail to accurately capture contact force interactions between objects in the packing space.

Currently, the only example in literature where object stability has been modelled accurately is in [55]. In this paper, authors ensure that each object is placed in a state of static equilibrium, accounting for gravitational forces and contact interactions with previously packed objects.

Whilst their work serves as an excellent first attempt at this problem, there are still several limitations to their work. Firstly, by limiting the placement of objects to locations where the object is in static equilibrium, it is possible that the algorithm may be unable to find such a location (in which case it terminates with failure [55]). Arguably, a better approach to this problem would be to prioritise placing objects in optimum locations (as found by the placement heuristic) and then simulating gravity and contact forces to allow the packing pile to settle naturally before adding the next item. Not only would this be more akin to real-world packing (where the stability of objects is difficult to know before placing them), such an approach would also enable a relaxation of the stability constraints, ensuring a 100%

success rate for object addition whilst also potentially allowing for tighter packing arrangements.

The second limitation with this approach is that, whilst static equilibrium may be achieved in the simulation, it does not necessarily mean that objects will behave this way in the real world. Force based simulations are inherently approximate by nature, since the accurate modelling of all physical interactions is highly challenging. As such, whilst they may give a good indication of stability in the real world, they cannot guarantee it. As an example, in [55], the authors validate their proposed methodology by attempting to recreate optimised packing plans using a robotic digital twin in a virtual physics-based environment. With a small number of objects (3-5 items), the robotic system successfully reproduced the packing structures with 100% success rate. However, as the number of objects increased (to 10), the success rate dropped to 80%, highlighting the increasing difficulty of replicating simulated packing structures as the complexity of the problem grows.

Given the difficulty with perfectly modelling physics-based interactions in virtual packing simulations, a more robust solution would be to integrate real-world feedback into the digital planning loop. For example, a packing algorithm could be coupled with a physical robotic system equipped with an overhead 3D scanner. After each object is placed into the container by the physical robotic system, the scanner would capture the updated status of the packing pile, which could then be compared to the digital counterpart to identify discrepancies. If discrepancies are identified between the physical and digital systems, the packing algorithm could then be re-run using the scanned packing configuration and remaining unpacked objects, allowing the system to dynamically adapt to any real-world deviations.

Manipulation Feasibility

An additional point which must be considered if packing structures are to be recreated in the real world is whether the desired packing locations are physically accessible (without collisions with other objects or the container boundary). The simplest way to achieve this is by placing the object in the desired location and then translating it vertically upward whilst checking for overlap.

Whilst this simplistic approach may be suitable for human packing, robotic packing must also account for the geometry and kinematics of the robot itself (to prevent collisions with both the environment and packed objects). As with stability, the only example in literature where this has been addressed is in [55]. Whilst also ensuring that objects are placed in static equilibrium, the authors also impose robotic manipulation constraints by simulating

a simplified top-down placement trajectory. This trajectory is explicitly modelled from the initial configuration to the final placement location, with continuous checking for inverse kinematic solvability, joint limit violations, and collisions.

The primary downside with their work however is that they only consider simple linear trajectories, and furthermore, they only model the robot's kinematic motion without simulating the robot's full dynamic behaviour. For example, they do not consider forces required to lift and move objects, and they do not explicitly simulate gripper interactions with the objects (i.e. how the gripper will grasp and release objects).

Capacity Constraints

In nuclear decommissioning, capacity constraints such as maximum allowable weight and radiation levels are fundamental to safe and effective packing. As such, containers are often designed with strict upper limits for both weight and radioactivity to comply with operational and regulatory requirements [102].

Regarding radiation limits, a key point to consider is the concept of self-shielding, where denser, or less radioactive, objects can block or absorb radiation emissions from more hazardous objects. For example, by placing more radioactive objects near the centre of the container and surrounding them with less radioactive or denser materials, it may be possible to shield external emissions more effectively. This in turn may allow for more radioactive material to be packed without exceeding eternal dose limits. One way in which this could be achieved is by using radiation dose estimation methods, such as the point-kernel approach (e.g. [103], [104]), which estimates radiation attenuation based on object geometry, material properties and placement.

Another key point to consider within this context is whether to treat radiation and weight limits as hard constraints or as objectives within a multi-objective optimisation framework. Treating them as hard constraints is the simplest way to deal with the problem, since they can be checked each time an object is added, with the packing process terminating when one of the limits is reached. Whilst this ensures strict compliance with limits, by not considering object weights and radiation levels in the optimisation process, it could lead to suboptimal packing solutions. With a hard limit approach, the risk is that one of the constraints could be violated before the algorithm has a chance to efficiently pack the container. For example, if the algorithm places many small but highly radioactive objects into a container, it is likely the radiation limit will be reached before the weight or capacity limit is reached, resulting in a poorly packed container. By distributing the small radioactive objects across containers, it could allow for much more efficient space utilisation.

To the best of the authors knowledge, there are no packing algorithms in literature which consider these factors. Whilst this is understandable for the majority of packing algorithms (as they are typically designed for non-nuclear applications), the limited examples in literature where packing algorithms have been developed for nuclear waste packing ([20], [21], [28], [81], [83]) do not consider these capacity constraint either, presenting an avenue for potential future work.

2.2.3.5 Rotation

In 3D irregular object packing, the rotation of parts plays a crucial role in determining how efficiently objects can be arranged in the packing space. In general, allowing objects to rotate freely (or using small angle increments) enables the objects to adopt a greater variety of unique poses, which in turn allows for more unique (and potentially better) packing layouts to be found. The primary limitation however is that this increases the search space, which in turn increases the computation time required to find solutions.

For example, in serial approaches (i.e. one by one packing), each time an object is added, the algorithm must search for a location to place the object. This typically involves fixing the orientation of the object and then searching over the packing space for valid locations where it can be placed (e.g. [23]). The object is then rotated by a fixed amount, and the scanning process must repeat for the new orientation. Once the scanning process has been completed for all allowed orientations, the algorithm can then select a site based on the choice of placement heuristic (e.g. the location which minimises the height field of the packing structure [23]). The problem however is that this scanning process must be repeated for all allowed object orientations, which grows exponentially with a decrease in rotation step size.

As a simple example, consider a case where object rotation is restricted to 90 degrees about each axis and another case where object rotation is restricted to 1 degrees about each axis. In the first case, the number of unique orientations the object can take is $4^3 = 64$ (4 orientations in 90-degree increments about each axis). In the second case, the number of unique orientations is $360^3 = 46{,}656{,}000$ (360 orientations in 1-degree increments about each axis). It is clear that, if the scanning process needs to be repeated for each allowed orientation, the number of times it would be repeated grows exponentially with a decrease in the rotation step size. The authors of [23] demonstrate this with their packing algorithm, producing the graphs shown in Fig. 2-13 below. It can clearly be seen that as they decrease

the rotation step size, the quality of the optimised solution increases (the bounding volume of the packed structure gets smaller), but the computation time grows exponentially.

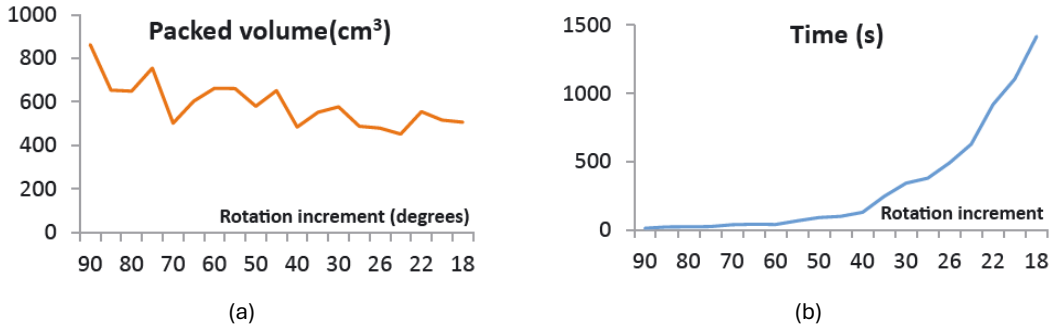


Fig. 2-13. Effect of rotation increment on quality of solution and run time. (a) – rotation increment Vs bounding volume of optimised packing structure. (b) – rotation increment Vs run time. Images from [23].

For one-by-one approaches where the orientation of parts is optimised by the high-level metaheuristic (rather than left to the placement heuristic to decide), the algorithm does not need to repeat the scanning procedure multiple times for each object (since the orientation for each object is pre-defined by the metaheuristic algorithm). The problem however is that by allowing objects to take on more orientations, the search space will grow, greatly increasing the time needed to find solutions which optimise both the packing order and the orientation of parts. The same problem also exists in parallel approaches, (e.g. where objects are drawn together from dispersed starting locations or where objects start with overlap which is iteratively resolved), since the number of allowed angles will directly affect the number of ways that objects can be translated/rotated at each iteration of the algorithm.

To combat this issue, most papers restrict the rotation of components (e.g. to 90-degree increments about each axis [58], [83], by only allowing rotation about one axis [82], [85], or by fixing the rotation of certain parts based on design requirements [65], [68]), or they do not allow rotation at all [31], [32].

Of all the algorithms presented in Tables 2-2 and 2-3, only a limited number allow continuous, or near continuous (small discrete steps), rotation of objects. For example, in [56], [60], authors use physics-based calculations to model object rotation and translations in the packing space, allowing object to rotate continuously when in contact with one another (to minimise intersection), and to try and align flat faces of neighbouring objects when they are not in contact (to try and maximise contact area between them when they are brought together). Similarly to [56], [60], in [30] authors present a physics based algorithm where objects are drawn together under simulated elastic forces. In this

algorithm, objects are allowed to rotate continuously, but *only* when they are in contact with one another. In [22], authors start with objects overlapping in the packing space and then iteratively calculate translation/rotation directions to remove overlap. In [81], objects are drawn together using simulated annealing to control travel/rotation step directions. In [47], objects are dropped from the top of the container one by one and are translated/rotated using biased random walks which favour downward motion. A point worth noting is that the final two algorithms ([47], [81]) do not allow true continuous rotation, instead restricting rotation increments to small values (1 and 0.1 degrees respectively).

It is also worth noting that in some cases authors use a two-stage approach, where an initial solution is generated using course finite rotation increments, followed by an improvement step where continuous rotation is allowed. For example, in [24], authors generate an initial structure where rotation is restricted to 90 degree increments, followed by a refinement step (where objects can translate/rotate continuously) to try and reduce the height of the packing structure further. In [59], similarly to [24], authors generate an initial layout with object rotation restricted to 90 degree increments. This is then followed by a shaking process which shakes the container up and down, allowing the objects to naturally settle into better packing arrangements. During shaking, objects can rotate/translate continuously, with their motion and interactions modelled using rigid body dynamics.

From this small selection, it is evident that the use of continuous rotation is very limited in packing literature, with most papers limiting rotation to increase speed. For one-by-one placement approaches, this is understandable due to the aforementioned problems of computational complexity and search space increase, which would make free rotation of parts impractical. However, for such approaches, there is a notable lack of research on placement heuristics which aim to exploit domain knowledge (e.g. geometric properties of the packed structure and object to pack) to select rotation angles intelligently. Instead, they either incorporate the orientations into the metaheuristic optimisation process or allow the placement heuristic to do a brute force search over the packing structure for all allowed angles (which are typically limited to large increments to increase speed).

To date, the only example in 3D irregular object packing where this has been attempted is in [84]. In this paper, the authors present a one-by-one packing algorithm for additive manufacturing where the packing problem is simplified to 2D by projecting the 3D objects onto a 2D surface. They utilise a novel angle selection heuristic to select a small subset of angles based on simple geometric properties of the objects (e.g., if the object resembles a circle then no rotation is permitted, if the object resembles a rectangle then 2 axis aligned

rotation angles are permitted). Whilst this study marks a good first attempt at trying to select rotation angles intelligently there are several limitations to their work:

1.) They simplify shapes to 2D – Whilst simplifying the packing problem to 2D allows it to be solved faster (and allows for easier exploitation of shape properties compared to complex 3D geometries), as discussed earlier, 2D projection methods are generally unsuitable for arbitrary 3D shape packing (e.g. waste packing) since it will likely lead to poor packing solutions, and makes incorporation of problem specific constraints (like object stability) much harder.
2.) Their shape analysis is simplistic – they only use a small number of basic shape descriptors which may fail to account for more complex geometries which often arise in 3D waste packing. Furthermore, they must manually define the rotation criteria for each geometric classification, rather than using an intelligent process to calculate desirable rotation angles automatically. As such this approach is more of a classification-and-selection strategy (which is sensitive to human initialisation), rather than a truly independent and intelligent process.
3.) No consideration for the packed pile or placement heuristic – Whilst they do consider the geometric properties of the next shape to pack, they do not consider the geometric properties of the packed pile or the choice of placement heuristic when deciding the rotation angles. As such it is difficult to know if the choice of rotation angles for objects is 'good' with regards to the current state of the packed pile and choice of placement rule being used.
4.) No validation of proposed heuristic – Following on from the previous point, whilst they propose a novel heuristic for calculating object angles, they do not compare their approach to a more conventional approach with fixed increments for all objects (e.g. 90 degrees). As such, it is difficult to know with certainty whether their approach does in fact improve the quality of packing over this simplistic approach.

Due to the aforementioned problems with this approach, the problem of how to intelligently select rotation angles in one-by-one packing methods remains an open challenge, with many avenues for potential future work.

2.2.4 Concluding Remarks: Review of 3D Irregular Object Packing

This review has explored the key strategies and challenges associated with 3D irregular object packing algorithms in literature, with particular focus on their application to nuclear decommissioning.

An in depth discussion has been provided on the strategies adopted by previous researchers for the packing of 3D irregular objects, examining how packing algorithms can build complete packing solutions from scratch (constructive approaches), and how the inclusion of metaheuristic optimisation algorithms (such as genetic algorithms) can be used to enhance the search ability of constructive approaches to find better packing arrangements (metaheuristic approaches).

Alongside this, the review also examined several key factors relating to the design of such algorithms, including how objects are represented (and how this affects the trade-off between geometric fidelity and computational efficiency), how packing strategies are adapted to different container types (i.e. bounded vs unbounded, single container vs multi container) and how real world packing constraints (such as stability and accessibility) are integrated. Additionally, the impact of object rotation on packing quality and computational complexity was discussed, highlighting the inherent trade-off which exists between the two and the need for more intelligent rotation handling strategies.

Based on this review, one-by-one constructive approaches emerge as being particularly well suited for waste packing applications due to:

- Their flexibility (both in terms of adapting the strategies to multi-container packing and ease of metaheuristic search integration),
- Their support for easier constraint checking (which can be performed as objects are sequentially added to the container),
- And their alignment with real-world packing practices (by explicitly defining packing sequences which can be replicated in practice).

Additionally, several areas are identified as promising areas for future work:

- **Consistent benchmarking** – one of the main limitations with existing packing literature is the lack of consistent benchmarking, with most studies choosing to use their own object datasets. As such, this makes the comparison of different constructive approaches and metaheuristics very difficult. Future work should focus on developing benchmark datasets for specific problem instances (e.g. multi container packing) to allow for better comparison between methodologies.

- **Hybrid metaheuristic approaches –** The existing metaheuristic approaches outlined in this review each have their own strengths and drawbacks. Given this fact,

combining metaheuristic techniques to try and leverage the strengths of different approaches (such as in [76] where a genetic algorithm was hybridised with a tabu search to prevent the algorithm revisiting past solutions), could help enhance their search abilities further. Given that the work in [76] is the only example where this has been done, hybridisation of techniques presents a promising avenue for potential future work. Furthermore, incorporating more advanced AI techniques, such as reinforcement learning, could help further enhance existing search techniques by allowing them to learn and dynamically adapt to different problems.

- **Allocation algorithms for multi container packing** – of the 52 packing algorithms presented in Tables 2-2 and 2-3, 45 of them (≈86.5%) focus on single container packing. Of the few multi-container approaches identified, all used incidental allocation, where the assignment of objects to containers is implicitly determined by the packing order. Future work should focus on exploring multi-container packing for 3D irregular objects in greater detail, with particular focus on developing prior allocation-based approaches (where the allocation of object sets to containers is determined prior to packing) to see how they compare against incidental allocation approaches.

- **Hyper heuristics for object placement** – One key limitation with one-by-one constructive approaches is that object placement is typically determined by a single placement heuristic (e.g. bottom-left-fill) which limits the number of potential placement locations for objects. Incorporating a hyper-heuristic framework (where the algorithm has access to multiple placement heuristics) could improve packing by offering more flexibility in terms of object placement. Of all the algorithms presented in this review, only one example exists (in [31]) where this has been done, highlighting the use of hyper heuristics as a promising avenue for future work.

- **Intelligent rotation angle selection** – As highlighted in the discussion section of this review, most packing algorithms tend to limit the rotation of objects to large increments (e.g. 90 degrees about each axis) to reduce computational cost (particularly for one-by-one placement approaches). While effective at speeding up placement location searches, this can limit the algorithm's ability to effectively fit objects together more tightly in the packing space. Despite this, there is a notable lack of research into intelligent rotation strategies which can exploit domain

knowledge (e.g. geometric features of the packed pile) to identify promising rotation angles, highlighting another avenue for future research.

- **Incorporation of real-world constraints** – A common limitation across existing packing algorithms is their lack of consideration for real world constraints such as object stability and placement accessibility. Whilst this lack of consideration may be acceptable in applications like additive manufacturing (where packed parts are printed in the packing space), such constraints are critical in waste packing scenarios, where the packing structure must be physically reconstructed. Future work should focus on integrating realistic physical constraints into packing algorithms, for example, by developing more advanced physics-based stability models or incorporating robotic manipulator path planning to evaluate location accessibility. In addition to this, research should also explore how to better interface digital packing algorithms with real-world systems to help minimise discrepancies between virtual packing plans and physical execution.

2.3 Cutting Optimisation in Nuclear Decommissioning

Whilst the packing literature forms the primary focus of this literature review, it is also necessary to consider cutting optimisation, particularly within the context of nuclear decommissioning. However, this section presents only a brief discussion rather than an in-depth technical survey. This is a deliberate choice driven by three factors:

- Firstly, the scope of this project is constrained by time and complexity and thus the decision was made to prioritise packing since packing is the limiting factor both from a computational perspective [22] (in large part due to computational costs associated with repeated intersection checks in packing) and from a cost perspective [15], [105] (primarily due to lifecycle costs arising from treatment, storage and disposal).
- Secondly, the decision was made early in the project to restrict cutting to planar cuts as they are more practical and tractable from an implementation perspective (with arbitrary cuts, there is an infinite number of ways to modify them, making the task of knowing which cuts to modify and how, very difficult).
- Lastly, whilst non-linear cuts may offer more flexibility, they also introduce additional modelling complexities beyond the scope of this project (such as dealing with interlocking parts produced by non-linear cuts).

As such, the following section will first present a brief overview of existing cutting optimisation algorithms in literature, including the limited examples developed specifically for nuclear decommissioning applications. Following this, a broader discussion will be presented highlighting several key practical considerations which arise in real-world nuclear decommissioning scenarios (such as factors affecting cutting cost, radiation exposure to workers, physical feasibility of part removal and structural stability during cutting), and how these considerations should inform the design of future cutting optimisation algorithms tailored for nuclear applications.

Broadly speaking, cutting algorithms may be split into two categories: surface segmentation and volume segmentation.

With surface segmentation, the goal is to partition the surface enclosure of an object. Such algorithms are often used for computer graphics tasks like texture mapping and animation [106]. Regarding nuclear decommissioning, the issue with surface segmentation methods are that they do not consider the objects internal volume when performing segmentation, i.e. they split the surface of the object into 'patches', but they do not volumetrically partition the model into distinct sub-parts. In reality, nuclear structures will be made of materials of varying thickness, or even solid volumes; any segmentation algorithm must account for this by segmenting the internal volume of the structure as well. Additionally, surface segmentation schemes do not produce individual cut parts (they only segment the surface of an object); if a packing optimisation algorithm is to be used to pack a set of cut parts (as is the case with this project), the cutting algorithm needs to produce individual cut parts which can then be packed. As such, surface segmentation schemes are omitted from this review section.

In contrast to surface segmentation, volume segmentation schemes aim to segment an object volume into separate sub-volumes. Such algorithms are often designed for segmenting parts for additive manufacturing. For example, in [107] the authors propose a volume segmentation algorithm which aims to partition a 3D input model (for 3D printing) based on predefined surface features (such as different coloured regions or regions made of different materials). One of their requirements for segmentation is that the cut parts must be assemblable, meaning there must exist a sequence of collision-free linear part trajectories that allows moving the parts away from each other until they are no longer in contact, or vice versa, moving them from disjoint positions back into assembly [107].

In [108] the authors propose an algorithm which aims to decompose a 3D object by analysing the surface curvature of the input model. The goal of the algorithm is to partition

objects for 3D printing such that the partitions facilitate ease of final assembly. The algorithm works by first analysing the surface curvature of the model to identify and extract ‘feature edges’ (edges shared by two faces whose normal vectors make an angle greater than a certain threshold). The edges are then grouped and filtered to produce a set of ‘feature loops’ (loops in the model where partitioning can occur). The algorithm then identifies the ‘best-fit loop’ and performs a partition at this point.

In [109] the authors propose two decomposition algorithms for the application of 3D printing based on planar cutting. The algorithms are: Multi-Objective Genetic Algorithm (MOGA) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES). MOGA is designed to simultaneously evolve solutions with different topologies and is able to find a final set of decompositions at variable complexity (i.e. solutions with a different number of pieces). In contrast, CMA-ES evolves a population of solutions with identical topologies and can find a limited number of comparable decompositions (solutions with the same number of cut parts) with significantly less computational time [109]. Both approaches utilise an evolutionary approach where each population member represents a cutting pattern which can be applied to decompose the object into multiple sub-parts.

In [18], the authors propose a genetic algorithm powered cutting algorithm for cutting nuclear structures, where the goal is to minimise tool wear, cutting time and scrap material. The algorithm is specifically tailored for cutting cylindrical objects (like reactor pressure vessels) and discoid objects (flat circular objects like a reactor core plate). The algorithm uses planar cuts, and for both cylinders and discoid objects, works in two stages. The first stage considers optimising cuts along one axis (y-axis for cylinders, x-axis for discoids) and the second stage optimises cuts along a second axis (xz-plane for cylinders, z-axis for discoids).

In [19] the authors propose a segmentation algorithm for the segmentation of a spallation target in the China initiative Accelerator Driven System (CiADS). The objective of the cutting algorithm is to cut objects into different sizes based on the distribution of radiation doses on the structure. The algorithm works in three steps: in the first step, the tetrahedral mesh of the object is converted into a voxel representation, then, a genetic algorithm is used to find non-linear segmentations for the object, and finally, the voxelised object with the corresponding cutting paths is converted back into a tetrahedral mesh representation.

As a point to note, the two examples of cutting algorithms outlined here are, to the best of the authors knowledge, the only cutting optimisation algorithms developed specifically for nuclear decommissioning applications.

2.3.1 Discussion

Decommissioning nuclear facilities is an expensive and complex process and as such, the optimisation of cutting plans for nuclear decommissioning must bear several key objectives in mind. The primary factors which should be considered are: minimising overall cost, minimising radiation exposure to workers, avoiding cutting through undesirable regions and ensuring the feasibility of a proposed cutting plan. Each of these factors are discussed in more detail below.

2.3.1.1 Minimising Cost

Minimising cost is one of the most important goals during the planning stages of nuclear decommissioning. The costs associated with the decommissioning of a nuclear plant is very high (often in the hundreds of millions per plant [110]); as such, one of the main ideas behind planning a cutting strategy is to find the most efficient way to dismantle structural components so as to lower costs. Costs are typically incurred in three ways; the type of tools used for a job, the total amount of cutting which is carried out and the number of cut pieces generated.

Efficient tool selection

Regarding tool usage, it is essential to consider the type of material and the thickness of the material being cut so that the right tool can be selected for the right materials. However, the problem with using different cutting tools is that some tools cost more to operate than others. For example, thermal cutting tools, such as plasma cutters can cut metal quickly, but also consume large amounts of energy and produce hazardous fumes/aerosols which require special ventilation or filtrations systems [3], adding significant cost and complexity. In contrast, mechanical cutting tools (such as saws, shears or nibblers) generally require less energy to operate and produce far less secondary waste (such as aerosols), reducing the need for more costly dedicated waste collection systems (e.g. filtration systems) [3].

As the majority of cutting algorithms (such as [107-109] outlined above) are designed for additive manufacturing purposes [111], the lack of consideration for cutting tools is understandable as no physical cutting is necessary, however the two algorithms designed for nuclear applications do not consider this either. The authors of [18] state that their algorithm considers tool wear, however they do not consider the use of specific tools (i.e. the user does not have the option to specify different tool types) and furthermore, it only

minimises tool wear by minimising the amount of cutting which must be done (i.e. the goal is to minimise cutting, which indirectly leads to less tool wear).

In [112] and [113], the authors present a virtual cutting environment which allows users to plan how to cut a structure and then performs the cutting using virtual tools. The algorithm allows the user to input a desired cutting plan and then select from a range of different cutting tools. The algorithm then applies the user-defined cutting plan to the structure in a virtual environment (using simulated robotic cutting tools) and shows the segmented structure along with the cutting costs, waste materials from cutting, and the mechanical properties of the waste products (such as volume and weight). This software is an excellent example of a digital twin, allowing the user to carry out a decommissioning scenario in a virtual environment, but the main drawbacks are that it requires a user-defined cutting pattern, and it requires the user to define what tools to use. As such, it cannot optimise a cutting pattern to minimise the use of expensive tools.

Going forward, an ideal cutting optimisation algorithm should incorporate information about the structure in question (such as the type and thickness of material present in the structure) and then use this to automatically select the most appropriate type of tool for different parts of the structure (e.g. using shears for thin pipe segments, using reciprocating saws for thicker steel components, etc), so that expensive tools (e.g. laser, plasma) are used only on sections where cheaper tools cannot perform effectively.

Reducing cutting length and part count

Apart from tool costs, the main factor affecting cutting cost will be the total amount of cutting required. Cutting through more material means more labour, more tool wear, more energy consumption and more secondary waste (e.g. swarf dust from metal cutting, aerosol from plasma cutting) that must be collected and disposed of. Therefore, the primary aim of cutting optimisation should be to minimise the overall cut length or volume. For example, in the cutting algorithm for nuclear decommissioning presented in [18], authors aim to minimise cutting by minimising the intersection volume between the cutting planes and the structure (which is analogous to minimising waste material, such as metal swarf, produced by the intersection of a cutting tool with cut material). In [19], the authors aim to minimise the length of the cuts applied to the surface of the structure whilst ensuring that the dose rate of each cut part remains within a user-specified limit. By tracking the cumulative length of cuts (or waste material removed by cuts), an optimisation algorithm can penalise solutions which involve excessive cutting in favour of solutions which achieve part separation with shorter cut paths (or reduced waste material).

Closely related to this is the minimisation of the number of cut parts produced. If a structure is cut into many small pieces, it will increase handling and packaging costs since each part must be cut, retrieved, and then placed into containers, which increases labour time and hence, costs. If possible, cutting plans should also aim to produce larger cut sections which can be removed in in one piece, rather than requiring division into many smaller pieces. Given the nature of this study (i.e. cutting *and* packing), in practice, this means a balance must be found: cut parts should be small enough to allow efficient packing into containers and to be handled safely given their weight/radioactivity, but not so small that it creates unnecessary amounts of labour.

Currently, the two cutting algorithms developed for nuclear applications ([18] and [19]) do not consider the number of parts, however, as with cutting length, this is an objective which can easily be built into such algorithms by tracking the number of parts and favouring solutions which both the number of parts and the amount of cutting required (for example, using a multi-objective optimisation framework to find the optimum trade-off between the number of parts and cutting length).

2.3.1.2 Minimising Worker Radiation Exposure

Another key consideration in nuclear dismantling is keeping radiation exposure to workers as low as reasonable achievable (ALARA) [114]. Even with remote tools and protective gear, human operators often need to be involved in cutting and packing.

Before a structure can be cut and packed, it may undergo decontamination to remove radioactive surface contaminants (e.g. flushing pipes, tanks or heat exchangers with chemical agents to capture and remove any remaining radioactive materials [115]). However, even with decontamination processes, it is not possible to remove all traces of radiation from nuclear structures, since some components (particularly components near the reactor core, such as RPVs) may become radioactive themselves due to irradiation by neutrons [116].

As such, even with prior decontamination and the use of protective equipment for workers, performing cutting still has the potential to expose workers to radiation. Due to this fact, a cutting algorithm developed for nuclear applications should also seek to isolate regions of higher radiation so that they can be cut and removed early in the decommissioning process. By cutting and removing highly radioactive components first (e.g. a highly activated reactor core plate or internals), these parts can be transferred to shielded storage, substantially

reducing the ambient does rate to workers moving forward (allowing them to work in the environment for longer).

In practice, decommissioning planners often follow such ALARA principles. For example, the US Department of Energy recommends performing as much work as possible in low dose rate areas by removing radioactive components to areas with low radiation levels before working on them [117]. With regards to cutting, this means that if a contaminated structure can be detached and then cut up in a workshop or behind shielding, this is more preferable when compared to performing extensive cutting in situ next to potentially other radioactive structural components. A cutting algorithm developed for plant-level applications (i.e. not just the consideration of a single component in isolation, but considering multiple components in a plant) could incorporate such considerations by scheduling cuts which result in the largest ambient dose reduction payoff (e.g. isolating and removing large structures first to reduce ambient does rates, before making additional cuts).

To date, the only cutting algorithm developed for nuclear applications which considers radiation dose rates for parts is [19]. In this algorithm, the authors utilise radiation mapping of the object and impose constraints on cut parts such that the dose rate of each cut part cannot exceed a user defined threshold. Whilst this is a promising first step, the algorithm only considers the radiation of the parts themselves without considering radiation exposure to the surrounding environment and workers. Future optimisation algorithms should aim to improve on this by incorporating methods to assess dose rates to the surrounding environment. For example, in [118] and [119], the authors propose a point-kernel based dose rate assessment algorithm for assessing the radiation dose of cut parts (produced by manually defined cutting schemes), and the radiation emitted to the surrounding environment. The algorithm also factors in the self-shielding properties (where the occlusion of a highly radioactive part by another less radioactive part blocks radiation) of materials to obtain more accurate dose rate estimations. This algorithm is an excellent example of dose rate assessment in virtual cutting (particularly because it factors in self-shielding), but the issue is that it requires a pre-defined cutting pattern, meaning it can only be used to evaluate a pre-existing cutting pattern, but it cannot incorporate the radiation data to optimise the cutting pattern.

2.3.1.3 Avoiding Undesirable Cut Regions

When planning how to cut a structure, there may often be regions or parts of the structure which should be avoided by cuts if possible. These could be regions which are highly activated (which may require isolation and removal as a single piece rather than cutting through it and risking the spread of contaminated material), or simply areas of a structure which are physically difficult to cut through or shouldn't be cut through (e.g. a pump housing system connected to the side of a tank, or support structures used to maintain structural stability, respectively). A cutting algorithm developed for nuclear applications should include constraints to avoid cutting through such regions. The authors of the nuclear cutting algorithm presented in [18] state that their algorithm is designed to avoid placing cuts through thick parts of structures, however this is only achieved by minimising the intersection volume between the cuts and the structure (which naturally leads to the algorithm avoiding cuts through thick regions). They do not however, explicitly enforce this constraint (meaning that cuts can still end up going through thick regions).

Implementing such 'no-cut zones' into an algorithm is relatively straightforward if the zones are known a priori (i.e. if there exists a map or labels indicating areas which shouldn't be cut). In the case of high radiation regions, this would require a radiation map of the structure (which could be used to identify high activity regions), or in the case of physical obstructions, an annotated CAD model of the structure. The cutting algorithm could then heavily penalise, or even disallow, any cuts passing through such regions.

The main challenge is that such regions must be known in advance. In real decommissioning scenarios, this means through characterisation: e.g. radiological surveys to map out contamination, and detailed engineering models or 3D scans to identify objects which should remain intact. This approach aligns with practical experience since human planners also make observational judgments such as "don't cut through this welded joint or this highly activated region; cut around it instead." Future work on cutting optimisation algorithm should therefore focus on ways to integrate information about the structure to cut (such as radiation heatmaps of 3D scans) to allow automation of such logic in the planning of cuts.

2.3.1.4 Ensuring Feasible Disassembly Plans

As with the packing of nuclear waste, another key point to consider when planning a cutting strategy using an optimisation algorithm is the feasibility of the cutting plan, i.e. can the plan be replicated in the real world? For feasibility, the main points to consider are the sequence

that parts will be removed (and whether each part can be removed from the structure without collision), and the stability of the structure during the disassembly (i.e. will the removal of a cut part affect stability of the remaining structure).

Sequence and Collision-Free Removal Planning

After making a cut, it is likely that the cut piece will require extraction (e.g. by a crane, robotic manipulator, or even a human operator). As such, there must exist collision free path that the part can follow such that it can be moved away from the structure and surrounding s without collision. Without consideration of this factor, an algorithm might propose cutting a large segment that cannot fit through available openings or is blocked by remaining parts of the surrounding structure. As an example, the cutting algorithm developed for additive manufacturing in [107] imposes feasible assemblability constraints by requiring that all the cut parts can be translated along linear trajectories out of the assembly without collision. Whilst the use of linear trajectories greatly simplifies the problem, it may not be suitable for any arbitrary structure. For example, in some simpler scenarios, complex path planning may be less of a concern. If a structure is in a large open area with ample space for crane access (e.g. an isolated RPV with an overhead crane mounted on a gantry), a feasible plan may be as simple as slicing the RPV into a few large ring segments and then lifting them vertically up and out [120]. However, for more cluttered environments (e.g. an RPV surrounded by pipes or other structures in the reactor hall), nonlinear trajectories may be required to extract parts without collision. In fact, material handling constraints (e.g. crane capacities, physical clearance of parts, or robotic manipulator reach) are among the most important practical limits in dismantling operations [5]. As such, plans must respect these limits by cutting and extracting parts such that they can be removed without obstructing narrow access passages or overloading equipment.

Closely related to this is the concept of removal sequence (i.e. what order parts be cut and removed from a structure) which will also affect collision feasibility. For example, a part which is initially obstructed by surrounding components may become free after another part of the structure is removed first. As such, the order of part removal is also another key consideration which should be factored into cutting optimisation to help avoid potential deadlock scenarios (where a part cannot be extracted).

As a final point, it is worth noting that not every decommissioning case will require careful consideration of disassembly sequencing and trajectories; in some cases, it may be acceptable to allow parts to simply drop after being cut. For example, during the decommissioning of the AT1 fuel reprocessing facility (La Hague, France), explosive cords

were used to cut sheet metal equipment in specially built hot cells [5]. The explosive cutting significantly reduced worker exposure (by minimising the time spent in the hot cells), but at the cost of producing scattered pieces and increased secondary waste (in the form of airborne particulate spatter) which needed subsequent collection and decontamination [5]. Additionally, during the dismantling of large concrete or masonry structures (such as chimney stacks or walls), the material of the structure may not lend itself to cutting into smaller, liftable sections. In such cases, it is often preferable to simply break the masonry and allow the rubble to fall (often inside the chimney stack or within an enclosure) to allow for subsequent collection and cleaning [121].

Such examples highlight the fact that not every decommissioning case will be a clean cut and remove scenario. If radiological conditions permit (or if it is necessary due to the structural material), allowing parts to drop and then cleaning up afterwards may be an acceptable strategy.

As such, consideration of such constraints when planning a cutting strategy is more essential for cases were allowing parts to drop may be impractical due to:

1. Safety reasons (e.g. allowing heavy cut segments from an RPV to drop, which could damage operators or the surrounding environment),
2. The risk of uncontrolled spread of contaminated material (e.g. if not performed in a controlled environment like a ventilated hot cell),
3. Or where dropping of parts may prevent further access to the remaining structure (e.g. if robotic manipulators are used to perform cutting in a constrained environment).

Structural stability during dismantling

Another key consideration regarding feasibility is ensuring that the remaining structure stays stable as parts are cut and removed. In contrast to additive manufacturing (where stability isn't a concern since parts are printed after partitioning of the virtual model), nuclear dismantling must avoid instability to prevent collapses or structural failure during the cutting process. For example, removing large sections lower down in a structure could result in the remaining structure bending or collapsing under its own weight. Conventional demolitions practice reduces this risk by working in a top-down fashion (removing higher parts before lower parts to prevent top heavy configurations), however due to the structural uniqueness of reactor buildings, this may not always be possible in nuclear decommissioning [122].

In reality, it is often common practice to add supporting elements to structures before performing dismantling [123]. However, even in such cases, it is still essential to consider structural stability to ensure that: a.) parts of the structure connected to the supports are not cut and removed first, and b.) any remaining parts of the structure connected to supports do not place excessive strain on the supports which could lead to failure in the support system.

It is worth noting that both cutting algorithms developed for nuclear applications ([18] and [19]) do not consider stability (or disassembly sequencing), highlighting a gap in prior research. As such, future cutting algorithms developed for nuclear applications should focus on integrating disassembly sequence planning along with methods to calculate collision free paths for the parts (e.g. using path planning algorithms used to plan trajectories in robotics), and methods to assess structural stability each time a part is removed (e.g. using methods like finite element analysis or physics engines). In doing so, the algorithm could verify any cutting patterns produced to ensure that they are physically realisable. If a cutting pattern is found to be physically infeasible, the cutting planner could then either reject such a cutting pattern, heavily penalise it (in the hopes that the optimiser will be guided to produce patterns which are feasible), or try to repair the infeasibility (by directly modifying cuts to try and enforce feasibility).

As a final point, it is worth highlighting the fact that consideration of both these factors (disassembly sequencing and stability) become much more important if robotic systems are to be used for cutting and packing applications. In practice, humans already apply principles such as disassembly sequencing and ensuring structural stability in practice during the planning of decommissioning. However, with increased interest in the use of robotics for autonomous dismantling activities [124-126], it follows that such systems will require reliable methods for planning and executing physically realisable cutting sequences to help further reduce the dependence on human operator planning.

2.3.2 Concluding Remarks: Cutting Optimisation in Nuclear Decommissioning

This section has presented a discussion on cutting optimisation in the context of nuclear decommissioning, outlining both prior research and key practical considerations relevant to real-world dismantling scenarios.

Previous examples of how 3D object segmentation has been approached in literature was given (including the limited number developed specifically for nuclear decommissioning), followed by a discussion on several key factors that future nuclear-specific cutting algorithms must account for to be practically useful in real-world environments. These include:

- **Minimising overall cost,** not just by reducing the amount of cutting, but also by selecting cost effective cutting tools based on material and geometry.
- **Reducing radiation exposure to workers,** by isolating and removing highly radioactive components early to reduce ambient dose rates.
- **Avoiding undesirable cut regions,** such as activated zones or structural components which may be hard to cut through.
- **Ensuring physical feasibility of disassembly,** by accounting for collision-free removal paths and maintaining structural stability throughout the cutting process.

Based on this review, several future directions are suggested for the development of cutting optimisation algorithms tailored for nuclear decommissioning:

- **Tool-aware optimisation:** Integrating knowledge of cutting tool types and material properties to enable automated tool selection for different parts of a structure (with the goal of minimising the use of expensive tools where possible).
- **Radiation-informed planning:** Incorporating radiation data to allow cutting plans to isolate regions of high radioactivity so they can be removed first.
- **Constraint-aware segmentation:** Related to the previous point, using structural data (along with radiation data) to prevent the algorithm placing cuts through areas of high radiation, or components which are physically difficult to cut through.
- **Feasible disassembly sequencing:** Implementing mechanisms to check the extractability of a cut part and to ensure structural stability is maintained when parts are cut and removed. This point is especially important if robotics are to be used for remote/autonomous cutting.

2.4 Literature Review Conclusions

Given the limited research into the dual optimisation of cutting and packing, the purpose of this review was threefold:

1.) To examine the methodologies used to link the cutting and packing problem in the few cases in literature where this problem has been addressed and highlight limitations with the existing approaches.
2.) To explore in detail the methodologies used in literature to optimally pack 3D irregular objects as well as discussing key aspects relating to the design and implementation of such algorithms and how these relate to the real-world packing of waste.
3.) To examine the limited research on cutting optimisation algorithms developed for nuclear applications and provide a discussion on key design aspects which need to be considered when designing such algorithms.

The review on previous cutting and packing algorithms (Section 2.1) highlighted several limitations in existing approaches, namely, their heavy reliance on a good initial partitioning of the object, their limited flexibility in how cuts can be modified during the optimisation process, and their lack of multi container packing strategies.

The review on 3D irregular object packing algorithms (Section 2.2) highlighted one-by-one sequential placement approaches as the most appropriate for packing nuclear waste due to their explicit definition of a packing sequence (allowing the digital packing structures to be replicated in practice) and their support for easier checking of constraints such as object stability (which can be performed each time an object is added). Additionally, genetic algorithms were identified as the most used metaheuristic for optimising packing, owing to their design flexibility, strong search ability and suitability for handling discrete optimisation problems. Based on the review, several areas were identified for future research, including: the development of consistent benchmarking datasets, hybridisation of metaheuristics, pre-allocation strategies for multi-container packing, use of hyper-heuristics for placement decisions, intelligent rotation schemes, and more sophisticated incorporation of real-world constraints.

The review of cutting optimisation (Section 2.3) presented several examples of object partitioning strategies in 3D and discussed key considerations for cutting in nuclear applications. These included the consideration of different cutting tools in the cost optimisation strategy, the need to minimise radiation exposure to workers, and the feasibility of removing cut parts from their surroundings. From this, several research directions were identified, including: tool-aware optimisation, radiation-informed cut planning, constraint-aware segmentation, and disassembly sequencing with consideration of part removal feasibility and structural stability.

Given the wide range of areas identified for future work and the limited timeframe of this project, only a subset of these gaps will be addressed in this project. Specifically, this project will focus on the following research contributions:

1. **Developing a general framework for linking the cutting and packing problem:** Relating to research objectives 1 and 2 (Chapter 1, Section 1.2), the primary focus of this project is to develop a general framework for cutting and packing which aims to address issues with previous approaches by: making it less sensitive to initialisation, allowing for greater flexibility in the modification of cutting patterns during optimisation, and integrating multi-container packing.

Relating to objective 3 (Chapter 1, Section 1.2), the project will also investigate the following ways to enhance solution quality and real-world applicability of the algorithm:

2. **Hyper-heuristics:** Exploring the use of hyper-heuristic optimisation strategies to enhance the solutions quality in one-by-one packing approaches.
3. **Pre-allocation based multi-container packing:** Comparing pre-allocation based multi-container packing to incidental allocation methods to examine their differences in terms of speed and solution quality.
4. **Disassembly sequencing for cut structures:** Investigating different optimisation strategies for optimal disassembly sequencing while accounting for part accessibility and structural stability.

CHAPTER 3
METHODOLOGY

This chapter presents the proposed methodology for linking the cutting and packing problem to find the optimum trade-off between the two processes. The proposed methodology is then implemented in 2D and tested in Chapter 4 (which covers all the 2D work conducted in this project). Based on the results from the testing work performed in 2D, an updated methodology is then presented in Chapter 5 for the 3D implementation of the algorithm.

Before implementing the final version of the algorithm in 3D, the decision was made to perform testing in 2D rather than 3D for two reasons. Firstly, reducing the dimensionality of the problem from 3D to 2D has the effect of greatly reducing the search space (e.g. by reducing the number of sites the packing algorithm must search over, and reducing the number of variables required to represent cuts), allowing ideas to be tested faster. Additionally, the 2D work was implemented primarily in MATLAB, which includes ready-made functions for 2D polygon modelling (most notably, the 'polyshape' library). This library allows 2D shapes to easily be defined from a set of vertex coordinates and provides multiple useful functions for performing operations on them, such as Boolean set operations (union, intersection, subtraction), rigid transforms (translation, rotation, scaling) and geometric queries (area, centroid, bounding box), helping accelerate implementation. It is also worth noting that all 2D methodologies were deliberately designed to be directly transferable to 3D to ensure that any ideas tested in 2D could be implemented in the final 3D cutting-and-packing algorithm.

In addition to testing the proposed methodology for linking the cutting and packing problem, Chapter 4 also presents several works (also implemented in 2D) which were conducted with the aim of investigating ways to improve the cutting and packing processes used in the algorithm. These works are:

- **Hyper heuristics for container packing (Chapter 4, Section 4.2):** This study presents a novel single-container packing algorithm that uses a genetic algorithm to optimise the selection of placement heuristics. The framework is then extended to multi-container packing, where both the choice of heuristics and the packing order of objects are optimised, with the aim of minimising the number of containers required.

- **Allocation based multi-container packing (Chapter 4, Section 4.3):** This study implements a prior-allocation strategy where the object set is partitioned and assigned to containers before packing. This is compared against the hyper-heuristic order-optimisation method, where the allocation of objects is incidental to the packing order. The goal is to primarily to assess the speed and effectiveness of prior allocation versus incidental allocation.
- **Disassembly sequencing for cut parts (Chapter 4, Section 4.4):** This study explores several novel strategies for optimising the disassembly sequence of a randomly cut 2D structure. The objective is to ensure that parts can be removed without collision and that structural stability is maintained throughout the sequence.

The motivation for conducting these studies was as follows:

- **Hyper-heuristics:** The packing strategy used in this project (described later in this chapter) uses a one-by-one placement approach where object locations are determined by a placement heuristic. Given this structure, combining multiple placement heuristics into a hyper-heuristic framework presented itself as a natural extension for improving packing quality. Additionally, the literature review (Chapter 2, Section 2.2) highlighted that hyper-heuristics have received relatively little attention in the context of irregular packing, presenting an opportunity for novel contribution.
- **Multi-container packing:** The motivation here was two-fold. Firstly, the packing method used in this project initially lacked support for multi container packing. Whilst this did not limit the ability to test the proposed methodology for linking cutting and packing in 2D, extending the algorithm to 3D required multi-container packing to better reflect real-world decommissioning (where large structures are typically cut and packed across multiple containers). Second, as highlighted in the review of previous cutting and packing approaches (Chapter 2, Section 2.1), none of the existing algorithms included multi-container packing, representing a clear gap in literature and opportunity for novel contribution.
- **Disassembly sequencing:** Similarly, the investigation into disassembly sequencing was motivated by two factors. As discussed in Chapter 2 (Section 2.3), existing literature on cutting optimisation for nuclear decommissioning fails to address the feasible disassembly of cutting plans. Furthermore, whilst robotic systems are beyond the scope of this project, there is growing interest in the use of robotics for automating decommissioning tasks. In future autonomous cutting and packing

applications, algorithms like the one developed in this project could serve as a foundation for such systems, allowing them to plan the task with minimal human intervention. In such cases, ensuring that cutting patterns are physically realisable will be essential. This study therefore explores methods for evaluating feasible disassembly, laying the groundwork for more realistic and automation-capable cutting plans in future.

As a final point to note, the detailed methodologies for these supplementary studies are not presented here for the sake of brevity. Instead, they are described in their respective sections within Chapter 4.

The remainder of this chapter is structured in three parts:

- **Section 3.1** defines the problem being addressed in this project, including how objects are represented, the cutting and packing strategies used (and how the cutting and packing costs are quantified), and the constraints considered in the optimisation problem. This provides the necessary context for the development of the proposed solution approach.
- **Section 3.2** outlines the proposed approach for linking the cutting and packing problem. It introduces the feedback-based optimisation framework used in this project, explains the rationale for formulating the problem as a multi-objective optimisation task, and describes the justification for using a genetic algorithm to drive the search.
- **Section 3.3** provides a detailed overview of the genetic algorithm framework. This includes a general introduction to GAs, an explanation of how the GA algorithm was adapted for this application, and a discussion on the choice of GA operators and parameter tuning strategies.

3.1 Problem Definition

This section outlines the formal definition of the cutting and packing problem addressed in this project. It begins by describing how input geometries are represented (Section 3.1.1), followed by the cutting strategy used and corresponding cost metric (Sections 3.1.2 and 3.1.3), and then the packing strategy and cost metric (Sections 3.1.4 and 3.1.5). Finally, relevant constraints and the strategy used to enforce them is discussed (Section 3.1.6).

3.1.1 Representation of Objects

For the 2D version of the algorithm a combination of both mesh and voxel representation are used. The shape to cut is defined using mesh representation (given as a set of vertex coordinates with connecting edges), with the cutting operations performed on the mesh representation. For the packing of parts into the container, a voxel representation is used (where shapes are approximated as a collection of small cubic units). Given that the packing algorithm operates in discrete voxel space, the mesh models for the cut parts were first voxelised before being packed by the packing algorithm, after which, the packed structures were then converted back to mesh representation for visualisation.

The decision to use this dual representation in the 2D version of the algorithm was made primarily for practical integration-related reasons. The packing algorithm used in this study (referred to as 'DigiPac', discussed in detail in Section 3.1.4) is a pre-existing packing algorithm developed in C++, which operates in voxel space and includes built-in functionality for voxelising mesh-defined objects. In contrast, the rest of the algorithm is implemented in MATLAB, which lacks native tools for the voxelisation of meshes. To bridge this gap between the two representations, the mesh-based shapes were defined and manipulated in MATLAB, then passed to DigiPac for voxelisation and packing. Once packed, the packing structure was converted back to mesh representation by DigiPac and then passed back to MATLAB for visualisation. This process greatly simplified the integration of the MATLAB code with the C++ based voxel packing algorithm, avoiding the need to develop a custom voxelisation algorithm within MATLAB.

This same dual-representation approach was also used for the 2D packing projects presented in Chapter 4 (I.e. the studies into hyper-heuristic and allocation-based packing). In each case, shapes were defined using mesh representation in MATLAB, then passed to DigiPac for voxelisation and packing, before being converted back to mesh format for further analysis and visualisation.

It is worth stating here that for the 3D implementation of the algorithm, a voxel-based representation was used throughout (both for cutting and packing operations). In this case, the input mesh models of the structures used for testing were converted to voxel format at the start, prior to being inputted into the voxel-based cutting and packing algorithm.

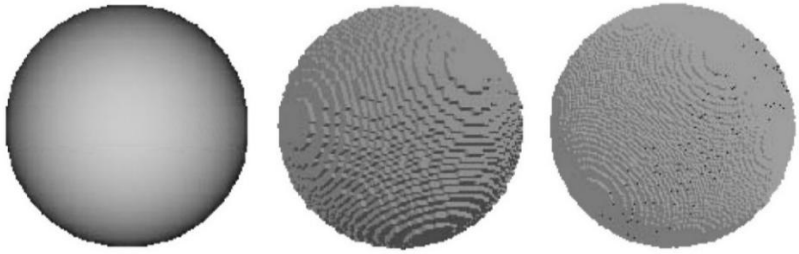
The reason for this was due to the fact that the 3D algorithm was implemented into the decommissioning software ‘NuPlant’ (developed by StructureVision Ltd. at the University of Leeds – [127]) which is written entirely in C++ and uses voxel representation by default (both for cutting and packing operations). The benefit of working exclusively with voxels within the

algorithm is that it removes the need for repeated format conversions during the cutting and packing process. Unlike the 2D implementation (where format conversion had to be performed every time the packing algorithm was called), the 3D version requires conversion to voxels only once at the beginning, reducing computational overhead.

The decision to use a voxel-based approach for 3D (and for packing in 2D) was primarily motivated by considerations of computational efficiency. Within the cutting and packing process, the packing stage tends to dominate the runtime due to the need for repeated intersection checks between complex objects. As discussed in the literature review, there is an inherent trade-off between the fidelity of shape representation and the speed of intersection checking during packing. Using simplified representations (such as 2D projection or bounding box approximation) will greatly increase the speed of the packing algorithm, but at the expense of poorer packing results due to an oversimplification of the shapes. Conversely, mesh-based representations provide more accurate modelling of object geometries, but are more computationally expensive due to the need to perform mesh-mesh intersection checks.

Voxel representation therefore offers a more practical compromise. By discretising the shape onto a regular grid of binary values (with occupied = 1, empty = 0) overlap checking is reduced to simple grid occupancy checks using fast array-based Boolean operations. This enables a relatively accurate approximation of object geometry (compared to simplified approaches) whilst still allowing for fast and efficient packing.

The primary downside of the voxel approach, however, is that there is no automatic way to determine the 'best' voxel resolution to use; this must be decided by the user on a per-problem basis. As with the choice of object representation itself, the main point to consider when deciding resolution is the trade-off between speed and fidelity of the input models. Using a course resolution will decrease the computation time but will negatively affect how well the voxelised model approximates the input mesh (e.g. Fig. 3-1).



(a) (b) (c)

Fig. 3-1. Example of voxelisation (a) – Mesh model of a sphere. (b) – Voxelised sphere with course resolution. (c) - Voxelised sphere with fine resolution. Images from [128].

For the work presented in this study, a ‘rule of thumb’ approach is adopted when selecting voxel resolution: use the lowest resolution possible that still preserves sufficient geometric detail, to maximise computation speed without significantly degrading the model quality. This rule of thumb approach is demonstrated in the following examples.

With the 2D implementation, the voxel resolution is tied directly to the size of the mesh-shapes as defined in MATLAB. For example, if a square is defined in MATLAB with (unitless) dimensions of 10x10, when voxelised, the dimensions will also be 10x10 voxels. As such, the voxel resolution is defined implicitly based on the scaling of the shapes in MATLAB. Referring to Fig. 3-2 as an example, when using a low voxel resolution (small shapes as defined in MATLAB), the voxelised shapes are unable to accurately capture the geometry of the input shapes. This leads to large digitisation error when converting the shapes back to mesh representation, which in turn leads to large gaps between the packed shapes in the mesh-based representation of the packing structure. In contrast, using a higher resolution captures the geometry of the shapes better, reducing the gaps in the structure when the shapes are converted back. The size of the shapes was therefore set to try and minimise the gaps without causing an excessive increase in the computation time.

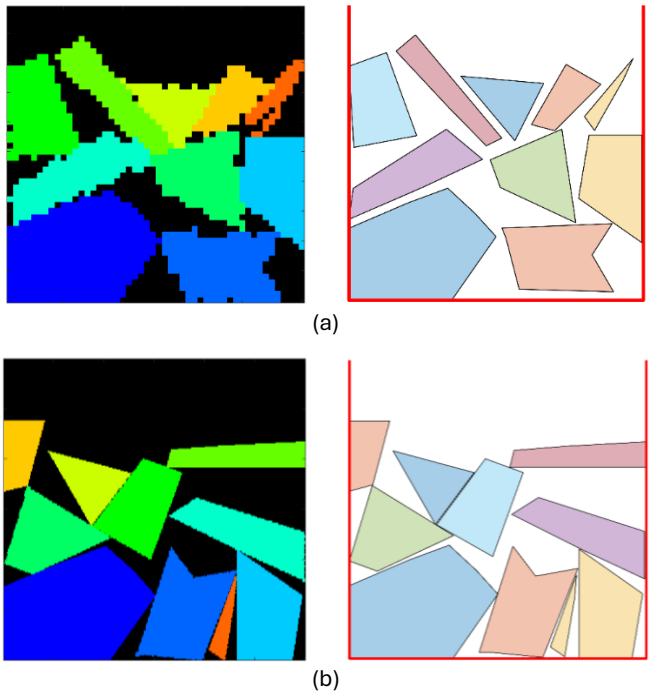
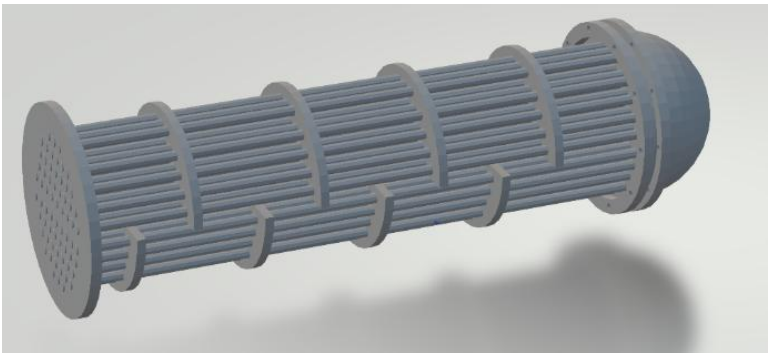
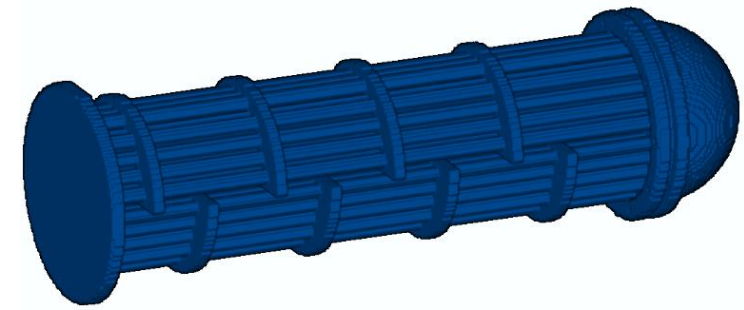


Fig. 3-2. Example of 2D packed shapes using different voxel resolutions. (a) – low-resolution results in large gaps in the structure when shapes are converted back to mesh. (b) – using a higher resolution reduces the gaps.

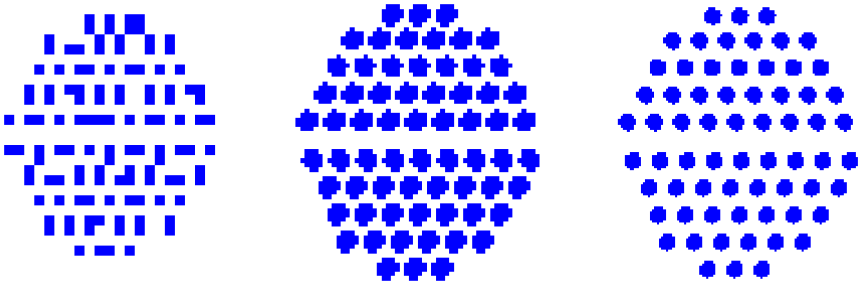
As with the 2D implementation, the voxel resolution of the models in 3D is also tied to the scaling of the input mesh model. Fig. 3-3 below demonstrates how the voxel resolution impacts geometric fidelity in 3D. Using too low of a resolution causes the pipe sections to merge during voxelisation (as can be seen in the cross-sectional view of the model), using a high resolution allows for better approximation of the input model at the expense of increased computational cost, the medium resolution offers a practical balance between the two.



(a)



(b)



(c)

Fig. 3-3. Voxel resolution for a heat exchanger model (a) – original mesh model, obtained from [100]. (b) – voxelised model. (c) – cross section of model for 3 different voxel resolutions: low, medium and high.

As a final point to note, it is important to acknowledge that the use of voxel representation comes with a particular problem which cannot be ignored. Whilst voxel-based representations help accelerate the packing process (by reducing the computational

complexity of overlap checks), they remain an approximation of the true real-world geometries regardless of resolution used.

In 2D, even at finer resolution, minor gaps or overlaps may still be present between shapes due to the voxelated nature of the representation. In 3D, increasing voxel resolution can reduce discrepancies between the voxelised model and the original mesh, but it cannot remove them entirely. Consequently, even high-resolution voxel models cannot guarantee a perfect match with the physical geometry of real-world components.

This discrepancy can have implications regarding attempts to physically replicate digital packing solutions in real world scenarios. For example, if the real-world object is slightly larger than its digitised counterpart, it may not fit into the intended location as defined by the packing algorithm. In contrast, if the digitised model is smaller than the real-world counterpart, it may introduce unintended gaps in the packing structure, allowing objects to shift which could compromise the stability of the packed pile.

It is worth noting that, whilst this is a major limitation with the voxel-based approach, it is not unique to it; all forms of digital representation suffer from approximation issues to some extent. For example, mesh models approximate geometry using polygonal facets with the number of such facets directly affecting how well a real-world object is approximated. Similarly, with point cloud scans, the number of points and level of noise in the scan will also affect how well an object is represented. Ultimately, what this means is that all digital representations involve trade-offs between accuracy and efficiency, and it must be accepted that no method can provide a perfect 1:1 recreation of real-world shapes.

3.1.2 Cutting Approach

As stated in the literature review, due to the challenges associated with non-linear cutting (In particular, the difficulty in knowing how to modify arbitrary cuts and risk of intractability arising from the vast number of ways in which arbitrary cuts can be modified) along with time limitations on this project, the decision was made to restrict cutting to planar cuts.

For cutting, a method called ‘Binary Space Partitioning’ (BSP) was selected. BSP works by recursively dividing the input shape into smaller parts [129]. Each cut is represented as a free-rotating straight line (in 2D) that splits an existing region, with subsequent cuts applied to the resulting sub regions to split them further. The cuts are constrained such that they do not intersect with existing cuts and are applied in a hierarchical fashion (Fig. 3-4).

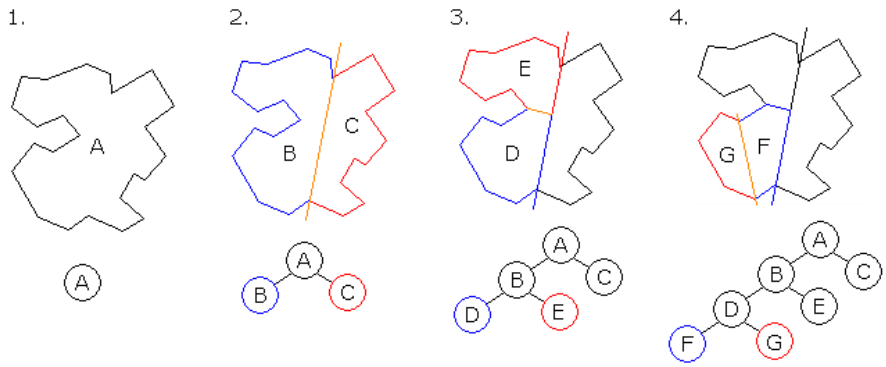
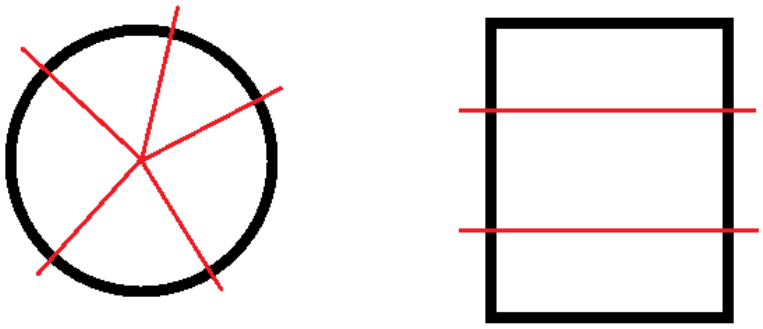


Fig. 3-4. Illustration of Binary Space Partitioning in 2D showing hierarchical structure. Image from [130].

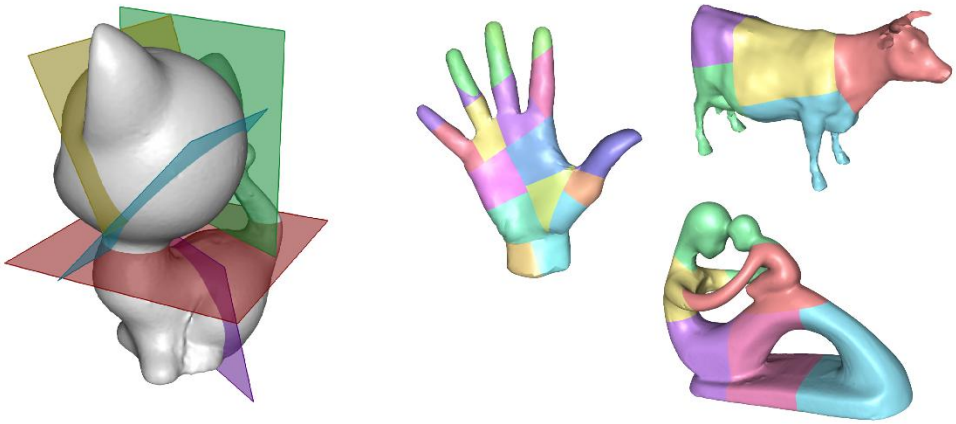
The decision to use BSP was motivated by a need for a general-purpose planar cutting scheme that was not tailored to a specific structure. For example, in the cutting and packing algorithm developed for RPV's ([28]), planar cutting was used, however the cutting logic was specifically designed around the cylindrical geometry of RPVs (by using radial and horizontal cuts as illustrated in Fig. 3-5).



(a) (b)

Fig. 3-5. Simplified illustration of RPV tailored cutting. (a) – Top-down view showing vertical radial cuts. (b) – Side view showing horizontal cuts.

In contrast, by using free-moving and free-rotating cuts, BSP provides a more flexible framework (which doesn't require assumptions about the input structure's geometry), making it suitable for a wide range of arbitrary shapes. As a point to note, BSP has been adopted in previous segmentation algorithms for additive manufacturing (e.g. [131], [132]) due to this exact reason. Fig. 3-6 illustrates examples from prior work where BSP has been successfully used to partition arbitrary input models, further demonstrating its versatility in real-world scenarios.



(a) (b)

Fig. 3-6. Examples of BSP cutting in additive manufacturing. (a) – 3D model showing BSP planar cuts. (b) examples of arbitrary shapes cut using BSP. Images from [132].

However, it is worth noting that this flexibility comes with a trade-off. Each BSP cut requires multiple variables to define it; an origin point and rotation angles about the centre point. In 2D, this would require 3 variables (x, y, θ), and in 3D, six variables $(x, y, z, \theta_x, \theta_y, \theta_z)$. This can significantly increase the number of optimisation variables when a large number of cuts is required, thereby expanding the search space. A larger search space means the algorithm must explore many more possible combinations to identify an optimal solution, making convergence slower. In contrast, with the aforementioned RPV cutting approach, each cut only requires a single variable; for radial cuts, an angle (θ) and for the horizontal cuts, a height value (z). This reduces the number of combinations the algorithm needs to consider, simplifying the search and allowing better solutions to be found faster.

As such, whilst BSP is adopted for the 2D testing of the algorithm, an alternative cutting strategy (referred to as 'orthogonal cutting') is also considered in the event that optimisation complexity becomes a bottleneck to performance. With orthogonal cutting, all cuts are axis aligned (i.e. orthogonal to the x, y, z axes) and allowed to pass through one another (Fig. 3-7). Previous cutting and packing research indicates that problems in which cuts are restricted to orthogonal directions are generally easier to solve than those with greater degrees of freedom (e.g. where cuts are allowed to rotate), due to the resulting increase in the size of the search space [24], [133], [134].

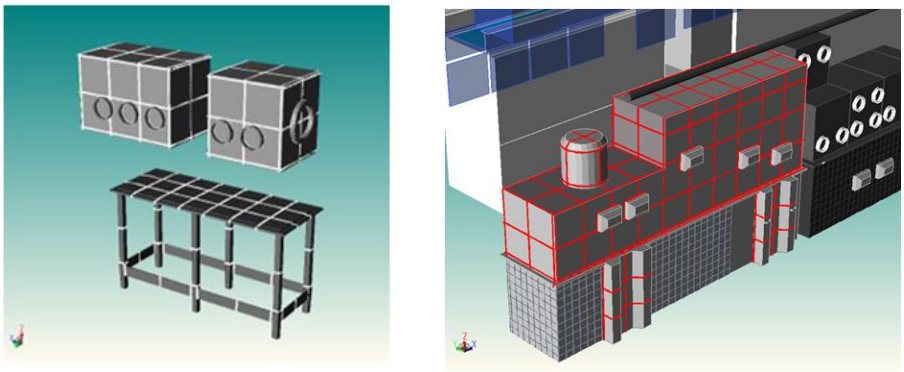
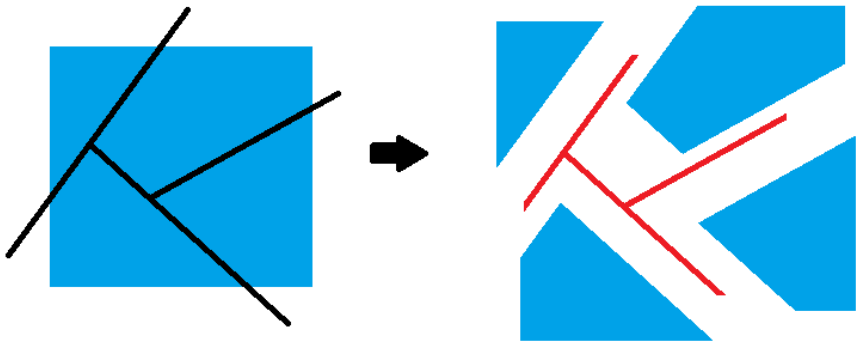


Fig. 3-7. Examples of orthogonal cutting applied to models of nuclear structures. Images from [135].

Whilst orthogonal cutting has reduced flexibility when compared to BSP, it is considered a compromise solution; it has more flexibility when compared to a structure-specific tailored approach, and it only requires one variable per cut (to control a cuts position along its respective axis) thereby reducing the complexity of the optimisation problem. If the BSP approach proves too complex during the 2D testing, the intention is to adopt orthogonal cutting as a fallback strategy for the 3D implementation.

3.1.3 Cutting Cost

The cutting cost of a cutting pattern is defined as the 'waste material' produced by all the cuts in the cut list. The cutting planes have a user-defined width and where the cutting planes intersect with the structure, the volume of the intersect region is taken as the 'waste' (Fig. 3-8). This is analogous to a cutting tool in the real-world producing waste material from the intersection of the tool with the material (e.g. a sawblade producing swarf in metal cutting). It follows that by minimising the waste material from a cutting pattern, the amount of cutting will be minimised.



(a) (b)

Fig. 3-8. Illustration of cutting cost. (a) – Structure (blue square) with cutting planes. (b) – Cut parts with plane intersection volume (i.e. 'cutting cost') shown in red.

The decision to use intersection volume as the cutting cost metric, as opposed to cutting length or cutting time, was made for several reasons:

- **Cutting length:** The problem with cutting length is that it does not give any indication of the difficulty with making cuts since it does not account the thickness of the material being cut through. For example, a 10cm cut through a 1mm piece of steel sheet will be much easier than a 10cm cut through a 200mm thick steel RPV wall.
- **Cutting time:** Whilst cutting time provides a more realistic measure of cutting effort, it introduces additional complexity. To estimate cutting time accurately, a conversion model would be required to translate geometric metrics (e.g. cutting length or intersection volume) into a measure of time. The problem with this is that estimating time is difficult due to its dependence of contextual factors such as the type of cutting tool used, the properties of the material and whether humans or robots are used to perform the task. Additionally, it can be argued that it adds a redundant step to the process, since a geometric measure of cutting (e.g. length or volume) is already required beforehand. It would be simpler, therefore, to work directly with this measure instead to avoid the need for a subjective conversion process.

In contrast, cutting volume (waste material) is both straightforward to compute and provides a geometry-based estimation of physical cutting effort, offering a practical balance between simplicity and realism. Additionally, this metric can be used consistently across a variety of structures and cutting patterns without requiring assumptions about tools, materials, or operators. Furthermore, if required for decommissioning planning, metrics such as the cutting length or time can be estimated as a post-processing step using technique-specific models, without needing to change the optimisation objective (i.e. intersection volume).

3.1.4 Packing Approach

In this study, the placement of objects into the containers is performed by the DigiPac packing software [47], [136], developed by StructureVision Ltd. at the University of Leeds. All packing methods in DigiPac use voxel-based representations for objects.

DigiPac offers several different methods for packing shapes into containers, which are:

- **Random Walks:** a stochastic method where objects are dropped from the top of the container one by one, with the motion of objects following random biased movements (rotations/translations). Downward motion is favoured (with upward motion only accepted with a small probability), mimicking a gravity based settling process.
- **Discrete Element Method (DEM):** A physics-based method where interactions are explicitly modelled using contact forces between objects. This approach produces more realistic and physically stable packing structures, but is very computationally expansive, making it impractical for cases where packing optimisation must be run many times.
- **Optimal Stacking Packing (OSP):** a one-by-one placement approach where objects are placed into the container one at a time, each time being placed according to the chosen placement heuristic.

For this study, the OSP method was selected for the following reasons:

1. Efficiency – OSP is much faster than DEM and is more predictable (and reliable) than stochastic random walks.
2. Practicality – OSP packing mimics real world packing by placing objects one by one. This process naturally lends itself to reproducibility in the real world (whether by robots or humans).

The process for packing an object using OSP is as follows. The algorithm first fixes the rotation angle of the object and then systematically translates it, in discrete steps, across each grid cell in the packing space lattice (Fig. 3-9 a), from the bottom of the container to the highest point in the existing packing structure (Fig. 3-9 c). At each discrete step, the algorithm performs an overlap check to see if the object overlaps with any of the previously packed shapes or container boundary (Fig. 3-9 b). If there is no overlap, the algorithm stores this site as a potential packing location. Once the entire lattice grid has been traversed, the object is rotated by a fixed angle increment (which is set by the user), and the shape is translated across the lattice grid again. Once the object has been translated across the packing space for all allowed orientations, the algorithm will search through all the stored feasible packing sites and select one based on the user-chosen placement heuristic (e.g. height minimisation which seeks to minimise the height of each object in the packing pile).

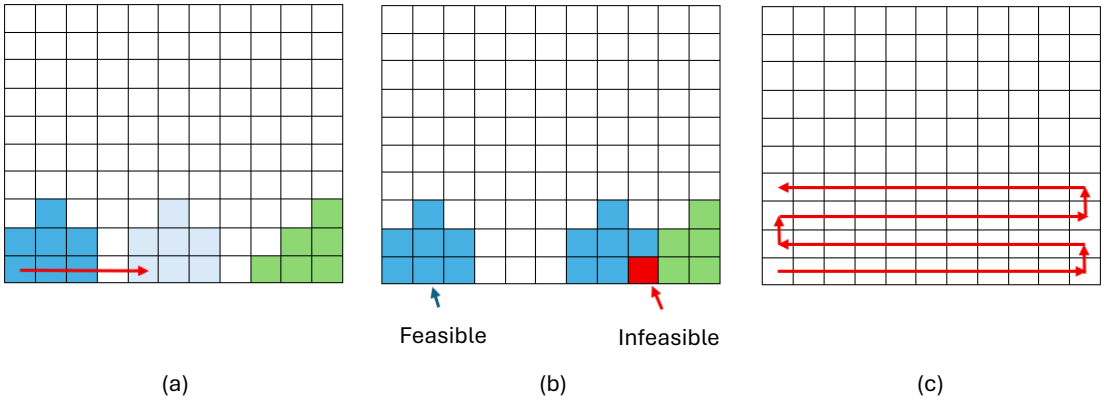


Fig. 3-9. Illustration of OSP packing. (a) – object to pack shown in blue, packed object shown in green. (b) – example of feasible and infeasible site (overlap shown in red). (c) – travel path for blue object in packing space lattice.

The motivation behind selecting DigiPac for the packing of objects was threefold. As discussed earlier, voxel-based representations simplify intersection checking, which is the primary source of computational overhead in packing optimisation [137]. Additionally, DigiPac is a commercially available software that has been developed and refined over many years, with multiple enhancements implemented to increase the speed of the algorithm (particularly with regards to overlap calculations) and reduce memory usage [138]. And finally, DigiPac also allows the use of arbitrary containers, rather than restricting the containers to cuboids (as is the case with previous examples of cutting and packing algorithms).

Regarding the choice of placement heuristic, the decision was made to use height minimisation for the 2D implementation of the cutting and packing algorithm. This decision was motivated by observations in the literature review, which identified height minimisation as a commonly used placement strategy. The idea behind height minimisation is that, by seeking to actively minimise the height of the packing structure, it promotes more compact packing arrangements. Additionally, whilst the algorithm developed in this project does not explicitly simulate dynamic stability (e.g. object movement under gravity or contact forces), it has been noted in literature that height minimisation indirectly promotes stability in packing structures by lowering the centre of mass of objects, reducing the risk of collapse due to imbalance [139].

It is also worth noting that, for the hyper-heuristic work presented in Chapter 4, additional placement heuristics were implemented and tested. More details on these placement heuristics and the rationale behind them are given in Chapter 4, Section 4.2.2.

As a final point to note, although the 3D version of the algorithm was implemented into the decommissioning software NuPlant, this software still utilised the DigiPac packing algorithm as the underlying packing process.

3.1.5 Packing Cost

As mentioned earlier, DigiPac initially lacked support for multi-container packing. Hence, for the 2D implementation of the cutting and packing algorithm, a single container with unbounded height is used for packing the shapes. As such, the packing cost in the 2D algorithm is taken as the percentage space utilisation of the container. This is calculated by dividing the volume of the packed objects by the volume of the packing space (defined as the width of the container multiplied by the peak height of the packing structure – Fig. 3-10). This gives a ratio between 0 and 1, with higher values indicating more compact packing solutions.

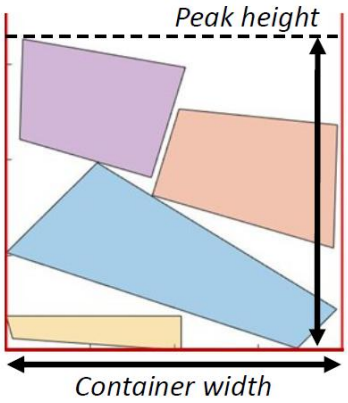


Fig. 3-10. Illustration of how cost is calculated for 2D packing.

As a point to note, a key focus of the research presented in this thesis is the extension to multi-container packing as this more accurately reflects requirements in nuclear decommissioning. Given this fact, the use of percentage space utilisation in the 2D case is considered more as a placeholder for packing cost until a multi-container strategy is implemented. Once implemented, the aim is to use the number of containers to represent packing cost, since the this is the primary factor in packing costs in real-world decommissioning (more containers means higher costs in terms of manufacturing, transport, and particularly, long term storage).

While percentage utilisation (in 2D) and number of containers (in 3D) are chosen for evaluating packing cost in this project, other metrics which were considered are:

- **Number of parts:** As noted in the discussion on cutting optimisation in nuclear decommissioning (Chapter 2, Section 2.3), the number of parts will impact cost by increasing the amount of packing which must be done. The problem with using the number of parts however, is that it is a poor indication of packing efficiency. For example, a small number of regular shaped objects (e.g. cubes) may pack together efficiently, whereas the same number of highly irregular objects may result in a far less compact arrangement.
- **Packing time:** The time taken to pack a set of objects may appear conceptually appealing (particularly for cases where operational time affects worker exposure), however as with cutting time, this is problematic as it would require detailed modelling of the packing process (e.g. motion planning for robotic manipulators or modelling of human handling times). Additionally, packing time can vary substantially depending on factors such as operator skill, available equipment and ease of container accessibility. As such, using packing time would be unsuitable as a general optimisation objective without significant additional modelling effort.

In contrast, percentage space utilisation (2D) and container count (3D) are straightforward to compute and align with real-world objectives (i.e. maximising space efficiency to minimise the number of containers required). Additionally, these metrics do not require assumptions about specific packing tools or operators and can be applied consistently across a wide range of packing scenarios and container configurations. Furthermore, as with cutting cost, if additional parameters such as packing time are required for decommissioning planning, these can also be estimated as post-processing steps from the final packing solution using technique-specific models, without requiring any change to the optimisation objective.

3.1.6 Problem Constraints

In the context of the integrated cutting and packing algorithm, several constraints (relating both to the cutting and the packing side of the algorithm) must be considered to ensure all solutions are physically realisable in the real world. Effective constraint handling is therefore essential to guide the optimisation process toward solutions which are not only optimised, but physically realisable as well.

Cutting Constraints

For the cutting side of the cutting and packing algorithm, a key constraint is to ensure that the cut parts produced by a cutting pattern are small enough to be packed into the desired containers. Even if a single cut part from a cutting pattern exceeds the maximum allowed size, the entire cutting pattern is considered infeasible.

Regarding the handling of this constraint, three common constraint handling techniques used in optimisation were considered:

- **Death penalty approach:** This approach is the simplest approach to dealing with constraint violation. With this approach, any solution which is infeasible (i.e. violates the constraint) is immediately discarded.
- **Penalty function method:** In contrast to the death penalty approach, rather than simply discarding infeasible solutions, the penalty function approach measures the degree of constraint violation and then adds a penalty term to the objective function (i.e. cutting cost) to degrade it. The idea behind this is that it allows better exploration of the search space by allowing partially infeasible solutions to remain, which could be made feasible with slight modifications.
- **Repair mechanism:** With repair mechanisms, rather than discarding or allowing infeasible solutions, the idea is to modify an infeasible solution to make it feasible (i.e. to enforce the feasibility constraint). Whilst such approaches can be highly effective, repair mechanisms are entirely problem dependent (meaning the user must devise a problem-specific method for modifying infeasible solutions to make them feasible).

For the 2D implementation, the penalty function method was adopted. The penalty for a given cutting pattern is calculated by iterating through all the generated cut parts and measuring how much any oversized parts exceed the container dimensions. These violation values are then summed and scaled by a user-defined penalty multiplier before being added to the cutting cost for that cutting pattern to degrade it.

The purpose of the penalty multiplier term is to control the severity of the penalty term by multiplying it to make it larger. This multiplier determines how strongly the algorithm is biased against infeasible solutions and will determine how aggressively the search is driven toward the feasibility region. Using a low penalty multiplier will lower the aggressiveness of the algorithm, allowing infeasible solutions to remain longer. This can help promote better exploration of the search space at the expense of longer convergence times. In contrast, using a high penalty multiplier will cause the algorithm to aggressively push solutions

toward feasibility, promoting faster convergence but with the risk of premature convergence to poor local optima.

The idea behind using the penalty approach is that by penalising infeasible solutions (rather than discarding them), the algorithm retains solutions which may be only slightly infeasible (which could be made feasible with minor adjustments). For example, a cutting pattern which produces only one oversized cut part could potentially be made feasible with only a small modification to the existing cutting pattern. Keeping such solutions in the population can be beneficial as it allows the optimisation algorithm to explore nearby regions of the search space that might contain high quality, feasible solutions. In contrast, using a simpler 'death penalty' approach (which immediately discards infeasible solutions) would cause such cases to be lost, potentially discarding promising search directions. Furthermore, once an infeasible solution is removed in the death penalty approach, it must be replaced, which is typically done by generating a new random solution. The issue with this is that there is no guarantee that randomly generated solutions will be feasible. If this process of removal and replacement happens repeatedly, the algorithm may spend many iterations generating and discarding solutions, effectively reducing the optimisation process to a random search and stalling meaningful progress.

As a final point to note, the repair mechanism was not selected primarily due to the complexity of implementing them. As stated, the problem with repair mechanisms is that there is no general-purpose approach, meaning that every repair mechanism must be tailored by the designer for the specific problem in question. Regarding the chosen cutting approach (BSP), the problem with repair mechanisms lies in the fact that it is difficult to know how many cuts would be needed (and where they should be placed) in order to segment an infeasible part to within the feasible limits.

However, it is also worth noting that if the fallback cutting method (orthogonal cutting) is adopted, a repair mechanism would be much simpler to implement. For example, if the distance between two consecutive cuts along an axis becomes too large, cuts can easily be inserted between them (e.g. Fig. 3-11). As such, in the event that the fallback cutting strategy is adopted, the plan is to implement this repair mechanism instead of continuing with the use of penalty functions.

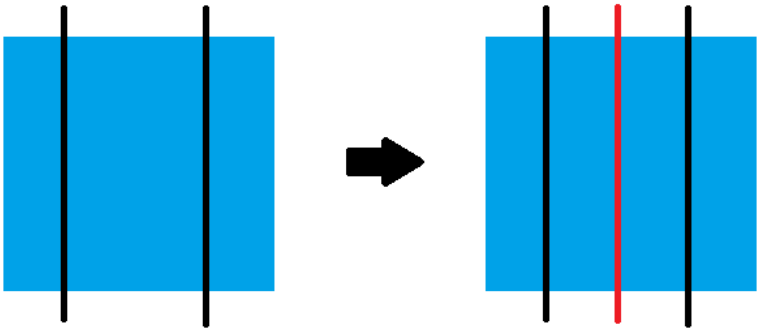


Fig. 3-11. Illustration of proposed repair mechanism for orthogonal cutting. When distance between two cuts becomes too large, cuts can easily be added between them.

Packing Constraints

For the packing side of the algorithm, the primary constraints are:

- No overlap between packed parts.
- All packed parts must be contained entirely within the container boundaries.

These constraints are treated as hard constraints and are enforced directly by the packing algorithm (DigiPac). During packing, when DigiPac translates a shape across the packing space lattice, it will only store candidate locations which satisfy these constraints. Any location that would result in a violation of these constraints is automatically discarded. As such, no additional penalty or repair mechanisms are required on the packing side.

3.2 Proposed Optimisation Approach

The goal of optimisation within the context of this project is to optimise both cutting cost and packing cost. As stated earlier, cutting cost is defined as the intersection volume between the cuts and the structure and packing cost is defined as the percentage space utilisation of the container. The aim, therefore, is to minimise cutting cost (i.e. reduce the amount of waste material produced during cutting) while simultaneously maximising packing efficiency (i.e. achieve high space utilisation within the container). The challenge, however, lies in the fact that these two objectives conflict with one another. Reducing cutting cost typically results in fewer and larger cut parts, which tend to pack less efficiently, leading to poorer space utilisation. In contrast, improving packing efficiency often requires breaking the object into smaller parts that fit more compactly, but this increases the total amount of cutting, thus raising cutting cost.

Because of this trade-off, the two objectives cannot be optimised independently. Solving one objective without considering its effect on the other can result in highly suboptimal solutions. For example, performing cutting optimisation in isolation to minimise cutting cost would likely produce large parts that are difficult to pack, leading to inefficient container use. Therefore, the problem must be approached as a multi-objective optimisation problem, where both objectives are considered together and solutions are evaluated based on how well they balance these competing goals.

The next two sections outline this proposed optimisation framework. Section 3.2.1 introduces the general principles of multi-objective optimisation, and the common strategies used to address such problems. Section 3.2.2 then presents the specific strategy proposed in this study to integrate cutting and packing within a single optimisation loop.

3.2.1 Multi-Objective Optimisation

When it comes to multi-objective optimisation problems with conflicting objectives (such as simultaneously minimising cutting and maximising packing efficiency), two primary approaches may be used to formulate and solve such problems:

- **Pareto-based (separate objective) optimisation:** In this approach, the goal is to identify a set of optimal solutions that represent different trade-offs between the objectives. This set of is referred to as the ‘pareto frontier’ and comprises multiple solutions to the problem where no single objective can be improved without degrading another. The resulting frontier allows decision makers to select a preferred solution after optimisation, depending on problem specific priorities.
- **Weighted objective function:** In contrast to the pareto approach (which treat the objectives as separate objectives), the weighted objective function approach instead combines the objectives into a single cost function, with each objective term in the function being multiplied by user-defined weights. These weights express the relative importance of each objective, allowing the optimisation algorithm to guide the search towards solutions that balance the trade-off according to these weights.

For the 2D implementation of the algorithm, the separate-objective pareto approach was used. The concept of pareto optimality is further illustrated in Fig. 3-12 below. In this simple illustration, each square represents a solution evaluated against two objectives, f1 and f2, where the goal is to minimise both objectives. Points A and B lie on the pareto frontier, with all solutions on this frontier being referred to as ‘non-dominated’ solutions. These solutions

represent the best trade-offs found. Referring to points A and B, neither A nor B can be improved in one objective without degrading the other; while point A has a better (lower) f2 value than point B, it has a worse (higher) value for f1, and vice versa. As such, neither A nor B is said to dominate one another (i.e. they are considered equally optimal).

In contrast, point C is said to be 'dominated' by both A and B because it is worse than both A and B in at least one objective without being better in the other (i.e. neither objective value for point C is better than for points A and B). As such, dominated solutions (like C) are considered suboptimal and not included in the final pareto set.

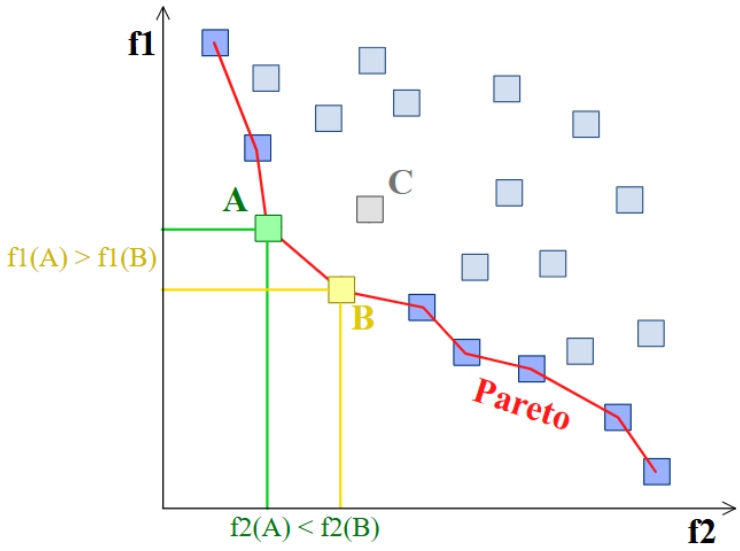


Fig. 3-12. Illustration of pareto optimality for a two-objective problem. Image from [140].

As such, the goal of multi-objective optimisation using this approach is to identify the pareto frontier (a set of solutions that dominate all other solutions in terms of their objective scores), thereby giving the user a set of optimal trade-offs from which a preferred solution can be selected based on problem specific priorities.

The decision to adopt this approach in the 2D implementation was motivated by 2 main factors:

1. **Simplified implementation:** The pareto approach allows the cutting and packing objectives to be optimised directly without the need to formulate a combined cost function. This simplifies the algorithm's design, particularly during early development stages, as it avoids the additional effort required to tune and validate appropriate weight values.

2. **Lack of clear cost weight for space utilisation:** In the 2D implementation, packing cost is defined as the percentage space utilisation of a container. Unlike discrete measures such as the number of containers (used in the 3D implementation), percentage utilisation lacks a natural and intuitive cost weight. It is unclear what a weight applied to such a ratio (e.g. 0.5 utilisation) would represent in real-world terms, making the formulation of a meaningful weighted cost function more difficult.

Given these considerations, the separate-objective Pareto approach was deemed the most practical and intuitive option for handling the 2D cutting and packing optimisation problem. This choice is also consistent with previous cutting and packing research, where Pareto-based multi-objective formulations have been used to explore trade-offs between cutting effort and packing efficiency without requiring predefined objective weights [28].

As a final point, it is worth noting that in the 3D implementation, a weighted cost function was used instead of the pareto-based approach. This change was made due to the use of multi-container packing in the 3D algorithm, which allows for the use of intuitive cost weights. For example, the cutting cost can be weighted by a unit cost representing the cost of removing a unit volume of material and packing cost can be weighted by the unit cost of a single container. This weighting scheme allows the computation of a single, interpretable total cost value for each cutting/packing solution, simplifying decision-making. A more detailed explanation of the cost formulation and the rationale for this change is provided in Chapter 5.

3.2.2 Approach for Linking Cutting and Packing

The optimisation of cutting and packing presents a unique challenge in multi-objective optimisation because the two objectives (minimising cutting and maximising packing efficiency) are sequentially dependent (i.e. packing cannot be performed until a set of cut parts has first been generated by the cutting process). This means that packing cost cannot be calculated from the cutting pattern alone, but only after the parts have been cut and then packed by the packing algorithm. As a result, conventional multi-objective solvers (which assume that all objectives can be evaluated simultaneously from a single candidate solution) cannot be applied to this problem.

Based on the review of previous approaches to the cutting and packing problem (Chapter 2, Section 2.1) two key limitations were identified with existing methodologies:

1. They have a heavy reliance on initial partitioning with limited flexibility to modify cuts during optimisation.
2. They do not all re-run packing optimisation from the start each time cuts are modified.

Due to the unique nature of this multi-objective problem and these identified limitations, a new methodology is proposed to explicitly link cutting and packing within this project.

The proposed approach solves the cutting and packing problem in a feedback-driven manner using a Genetic Algorithm (GA) to generate cutting patterns which drive down both the cutting and packing costs (Fig. 3-13). In this process, the packing algorithm is treated as part of the solution evaluation process, rather than as a separate process to optimise in tandem with cutting. That is, the outputs from both processes (cutting cost *and* packing cost) are used to assess how good a cutting solution is. The theory behind this is that, by using both the cutting and packing costs to optimise only the cutting patterns, the GA will guide the modification of cutting patterns such that it leads to good cutting solutions *and* good packing solutions (cutting patterns with minimised cutting which facilitate compact packing arrangements for the cut parts).

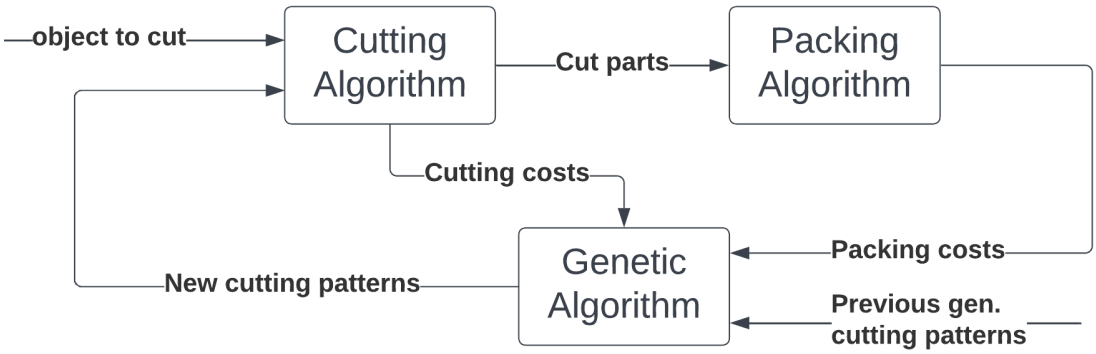


Fig. 3-13. Flowchart outlining the cutting and packing optimisation process.

The algorithm itself is a population-based algorithm, meaning that it works on a set of solutions (i.e. a set of different cutting patterns), rather than just a single solution. In each iteration of the feedback loop, the set of cutting patterns are evaluated by applying each one to the input object to decompose it into a corresponding set of cut parts. Each cutting pattern's associated set of parts is then packed by the packing algorithm, and the quality of each cutting pattern is assessed based on both cutting and packing costs.

The GA then generates a new set of cutting patterns via crossover and mutation operations, which aim to exploit the best-performing solutions (cutting patterns) in the current

population by combining and modifying them to discover potentially better configurations. This process allows the algorithm to explore a diverse range of solutions while steadily improving overall performance. Through successive iterations, the population evolves toward higher quality solutions, ideally converging on an optimum trade-off (where the amount of cutting cannot be reduced without degrading the packing, and the packing cannot be improved without increasing cutting).

The benefit of the proposed approach compared to literature is three-fold:

1. By using a population-based optimisation approach (instead of a single-solution approach), it reduces the reliance on a good initial partitioning schemes (since the population can be initialised with multiple cutting patterns) and allows the algorithm to explore a more diverse range of solutions throughout the optimisation process (reducing the likelihood of convergence to a poor result).
2. By using tailored crossover and mutation operators (which are discussed in greater detail in the subsequent section), the algorithm has the ability to add, remove and modify cuts in cutting patterns, increasing its flexibility in terms of how cutting patterns can be modified.
3. By running packing optimisation on every cutting pattern generated by the GA (rather than focusing on locally improving a single packing solution), it reduces the likelihood of the algorithm overlooking good packing configurations which could otherwise be missed.

The decision to use a genetic algorithm for solving the cutting and packing problem was based on a consideration of several alternative approaches, namely: 1.) simple search methods (such as brute force and random search), 2.) alternative metaheuristics (particularly single solution versus population-based approaches), and 3.) other population-based approaches aside form GAs. Each of these was evaluated for their suitability, as outlined below:

- **Brute force/random search**: Naïve approaches such as brute force search (systematically trialling every possible way to cut a structure) or random search (randomly generating cutting patterns) were quickly ruled out due to their inefficiency. The issue with such approaches is that, even when cutting is limited to planar cuts, the number of ways to cut a structure is vast. Trying to exhaustively evaluate or randomly sample cutting patterns would be computationally expensive and unlikely to yield high-quality solutions in reasonable time. In contrast, metaheuristic algorithms are designed to explore large and complex search spaces

in a more intelligent, guided manner. Rather than generating solutions blindly, they iteratively refine solutions by building upon the success of previously found good solutions. This structured search approach makes them a much more attractive alternative compared to simplistic methods like brute-force or random searches.

- **Population based vs single solution metaheuristics:** Single solution metaheuristics, such as the ones outlined in the 3D packing optimisation review (e.g. simulated annealing, local search algorithms and pattern search) were also considered but ultimately not selected. Such methods work by iteratively improving a single candidate solution (rather than a population of different solutions) and are more sensitive to initialisation, with a greater risk of premature convergence to poor solutions if initialised poorly. In contrast, population-based metaheuristics maintain and evolve a diverse set of candidate solutions, reducing their sensitivity to initialisation and enhancing their ability to better explore a diverse range of different solutions throughout the optimisation process. This makes population-based approaches more suitable for the highly combinatorial nature of the cutting and packing problem addressed in this study.
- **Genetic algorithm vs other population-based metaheuristics:** Among the various population-based approaches, genetic algorithms were selected due to their extensive track record in solving discrete, highly combinatorial and multi-objective optimisation problems. GAs are well suited for this work since they offer a high degree of modularity and customisability, which is particularly useful when designing problem specific crossover and mutation operators tailored to cutting patterns. In contrast, swarm-based approaches (such as the firefly algorithm outlined in the 3D packing review) were considered less suitable since they are typically designed for continuous search spaces (where variables are defined on a continuous range) and require additional mapping strategies to handle discrete problem domains (like the voxel-based cutting used in the 3D algorithm). Additionally, unlike GAs, they lack crossover mechanisms which limits their ability to exploit and refine good traits (e.g. good partial cutting sequences).

As a final point, it is worth acknowledging here that GAs were used extensively throughout this project, particularly in the 2D work presented in Chapter 4, due to these very advantages (i.e. reduced sensitivity to initialisation, good search ability and flexibility with implementation). Furthermore, as noted in the literature review, GAs are the most widely used metaheuristic in 3D packing optimisation, further supporting their suitability for such problems. Their strong track record in this domain, combined with the vast array of existing

literature on GA variants and enhancements, also make them an ideal foundation for future improvements regarding the work represented in this thesis.

3.3 Genetic Algorithm Framework

The following section outlines the structure and methodology for the GA used in this project, beginning with a general overview of the GA process (Section 3.3.1) followed by a detailed outline of the project-specific GA framework used to solve the cutting and packing problem (Section 3.3.2). The selection, mutation and crossover operators used in the proposed framework are then described in detail, along with the rationale for their use (Section 3.3.3), and the importance of key GA parameters (and the strategy used for tuning them) is discussed (Sections 3.3.4 and 3.3.5 respectively).

3.3.1 General Overview

Genetic algorithms (GAs) are a class of population-based metaheuristic optimisation techniques inspired by the process of natural selection and genetics. GAs have received much attention in literature over the years, being applied to many different complex, real-life problems in a wide range of fields (such as economics, engineering, politics and project management) [141]. In particular, they are also well suited for solving constrained multi-objective problems with large search spaces [142], such as the problem being addressed in this project.

In a typical GA, the algorithm maintains a population of candidate solutions (called ‘chromosomes’) which are evolved over multiple successive iterations (or ‘generations’). Each chromosome encodes a potential solution to the problem, and it is the fitness of each chromosome (i.e. how good the solution is with regards to the optimisation objective) which the algorithm seeks to improve over time. Fig. 3-14 below shows a flowchart for the standard GA optimisation process.

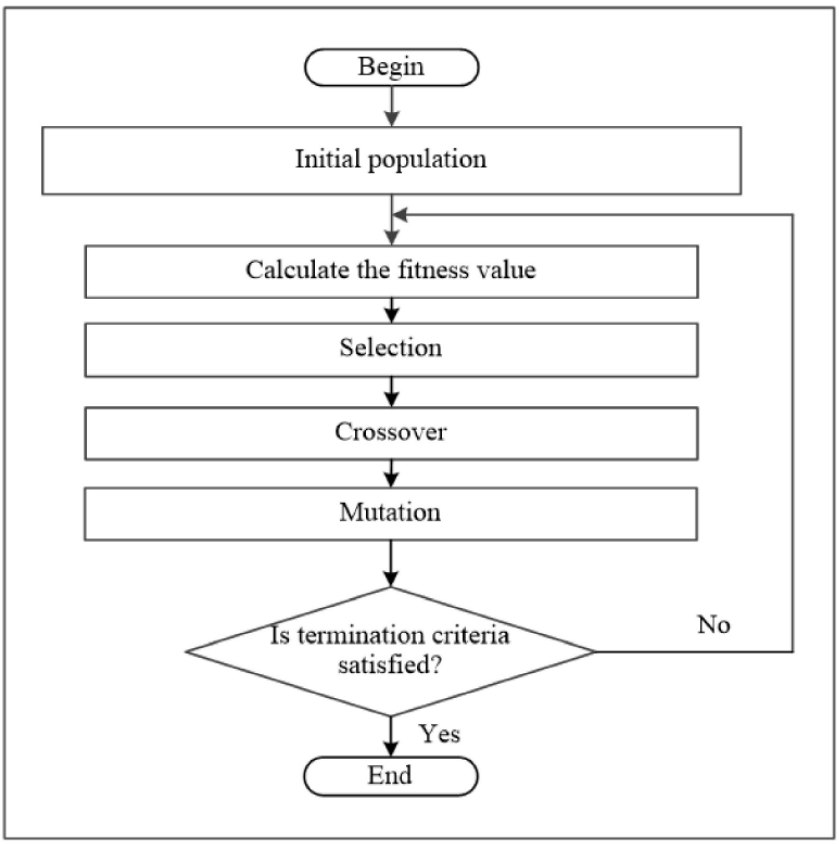


Fig. 3-14. Flowchart for a standard genetic algorithm. Image from [143].

The main steps involved in a GA are:

- **Selection:** Determines which chromosomes in the population get to reproduce. Typically, selection is designed to favour the selection of better performing individuals (i.e. those with higher fitness with regards to the optimisation objective), with the aim being to pass on good genetic material from these high performing solutions to the next generation. Common strategies for selection include roulette wheel selection, rank selection and tournament selection (discussed in Section 3.3.3 below).
- **Crossover (recombination):** Combines two parent chromosomes (obtained via selection) to produce two offspring. This process mimics the natural process of reproduction, with both offspring inheriting characteristics (or 'genes') from both parents. Various crossover methods exist in literature, such as single point, multi-point and uniform crossover (discussed in Section 3.3.3 below).
- **Mutation:** Introduces small, random variations to chromosomes to help preserve genetic diversity in the population and prevent premature convergence. This process is analogous to genetic mutation in nature, and typically involves altering one or more genes in a chromosome with low probability.

- **Replacement and Elitism:** When a new population of solutions is generated (via selection, crossover and mutation), this new population forms the next generation, replacing the population from the previous one. Additionally, GAs often employ 'elitism', where a small portion of the best solutions from the current population are carried over to the new population. This ensures that the best-found solutions are not lost as the population evolves.

It is through this iterative process that GAs are able to incrementally improve a diverse range of solutions, enabling them to explore large and complex search spaces (i.e. problems with many different combinations) in a directed and efficient way.

3.3.2 Project Specific Framework

The flowchart in Fig. 3-15 illustrates how the genetic algorithm works within the proposed optimisation framework.

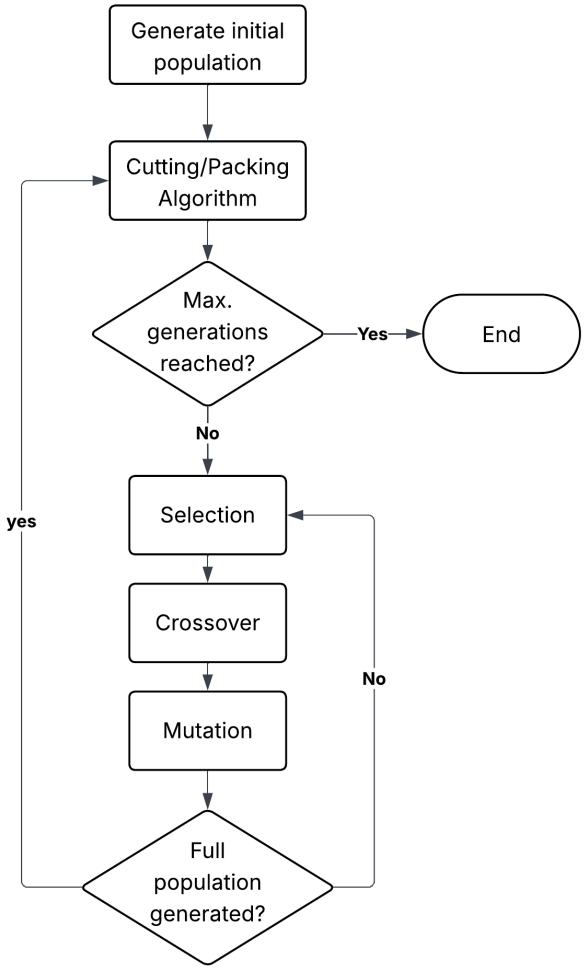


Fig. 3-15. Flowchart for the cutting and packing GA.

The GA begins with a randomly generated initial population of cutting patterns (with the population size set by the user) where each population member (chromosome) is a list of cuts, with each 'gene' in a chromosome representing a single cut.

Once generated, each cutting pattern is then applied by the cutting algorithm to the input structure to decompose it, and each set of corresponding cut parts are then packed by the packing algorithm. The algorithm then ranks the population of solutions from best to worst based on the cutting and packing costs. For the 2D implementation, since the problem is solved using a separate pareto approach (see Section 3.2.1), a pareto-based ranking is used where the population is ranked into successive pareto fronts (i.e. the first front contains all non-dominated solutions, the second front contains solutions dominated only by those in the first front, and so on), as shown in Fig. 3-16 (a). In the 3D implementation, since a weighted objective function is used, the ranking is instead based on the single value (i.e. total cost) returned from the weighted cost function (Fig. 3-16 b). These rankings (whether Pareto fronts or scalar cost) are then used to guide the selection of individuals for reproduction.

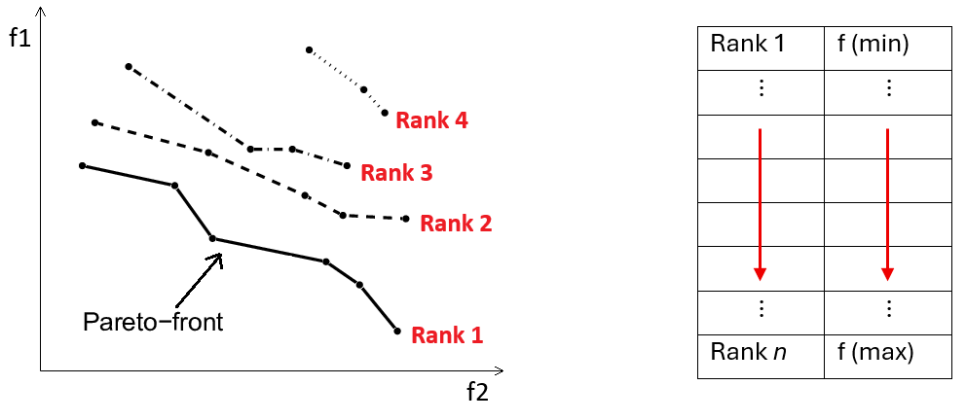


Fig. 3-16. Illustration of different ranking systems. (a) – Pareto based ranking (image adapted from [144]). (b) – objective score ranking.

Once ranked, the algorithm then applies selection, crossover and mutation to the population to generate the new child population. For selection, tournament selection is used where the algorithm will select a random subset of individuals (the tournament pool) from the population (with the tournament pool size set by the user). The algorithm will then select the best individual from the tournament pool as a parent solution (Fig. 3-17).

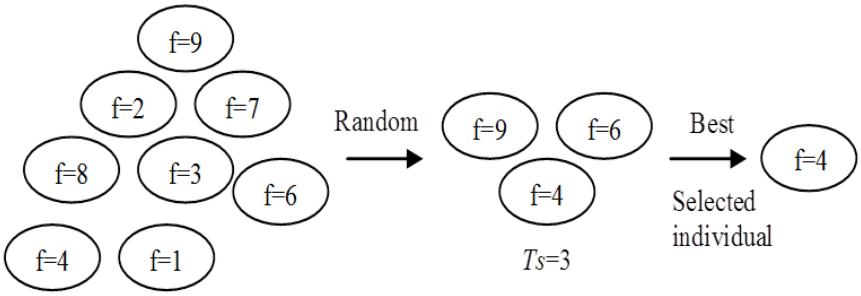


Fig. 3-17. Illustration of tournament selection with a tournament size of 3. Image adapted from [145].

When generating child solutions, tournament selection is run twice to select two different parents for crossover (i.e. the same parent cannot be selected twice). For crossover, random single point crossover is used, where both cut lists from both parents are split at random points. The first child solution is then created by combining the first segment from one parent with the second segment from the other parent. The second child solution is produced in the same manner, but with the parent roles reversed, i.e. the first segment from the second parent is combined with the second segment from the first. This process is illustrated in Fig. 3-18.

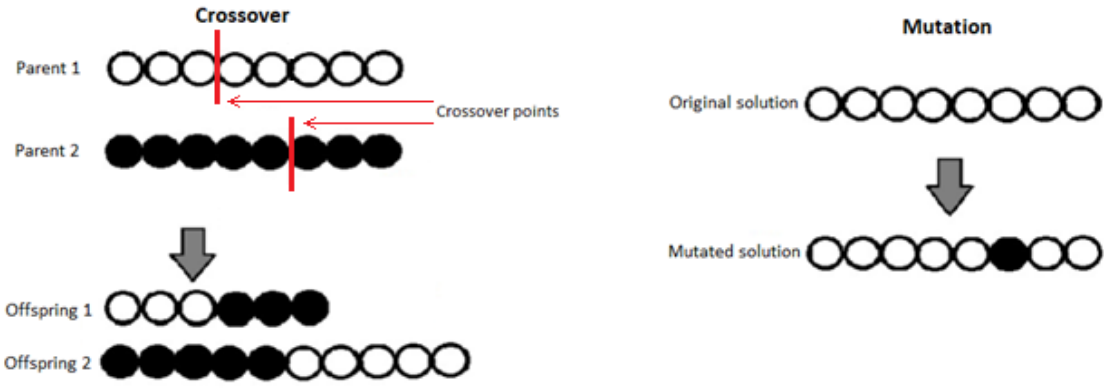


Fig. 3-18. Illustration of crossover and mutation. Image adapted from [146].

After crossover, the two child solutions are then subjected to mutation. During mutation, the algorithm will cycle through the list of cuts in a child solution, with each cut having a small chance of being mutated (with the chance of mutation set by the user). When a cut is mutated, the algorithm will apply a small random change to the cut to slightly perturb its position in space.

The process of selection, crossover and mutation will continue in a loop until enough new child solutions have been generated to fill the next generation's population. However, before this process is run, a small portion of the best performing individuals from the previous generation (the 'elite' population) are directly carried over to the next generation without modification (with the size of the elite population also being set by the user). Once copied over, the algorithm will proceed to generate the rest of the population via selection, crossover and mutation until the total population size (i.e. the elites + the newly generated child solutions) reaches the user-defined population size.

The algorithm then loops back to the start and the new population of cutting patterns is evaluated by the cutting and packing algorithm. The process of evaluating the population and generating a new one will repeat in a loop until the maximum number of generations (set by the user) is reached, at which point the algorithm will return the optimised pareto frontier (or in the 3D case, the solution with the lowest total cost).

3.3.3 Operator Selection

This section provides a justification for the selection, crossover and mutation operators used in the cutting and packing GA, along with a discussion on some commonly used alternatives.

Selection Operator

When it comes to parent selection in GAs, the three most commonly used methods are:

- **Roulette wheel selection:** Each individual in the population has a probability of being selected based on its fitness value, with fitter individuals having a proportionally greater chance of being selected (Fig. 3-19). To select a parent, the wheel is spun and the individual whose segment the pointer lands on is chosen. This method works directly with raw fitness scores and is simple to implement but can suffer from issues if the fitness values are very similar (resulting in nearly equal segment sizes, reducing selection to near random selection), or if the population is skewed by outliers (where exceptionally fit individuals dominate the roulette wheel and are almost always selected) [147].

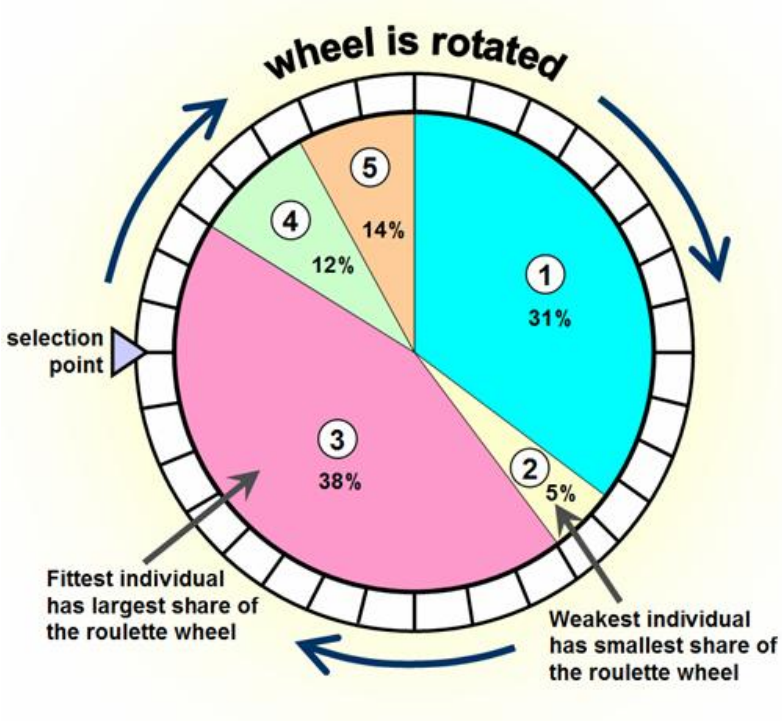


Fig. 3-19. Illustration of roulette wheel selection. Image from [148].

- **Rank based selection:** Similar in principle to roulette wheel selection, except the selection probabilities (segment sizes in the roulette wheel) are assigned based on an individuals rank rather than raw fitness (e.g. rank 1 for the least fit, rank *n* for the most fit in a population of *n* individuals). This method reduces sensitivity to outliers and populations with similar fitness scores, but requires a mapping scheme (e.g. linear or exponential scaling) to convert the ranks to probability values [147].
- **Tournament selection:** A group of individuals (the 'tournament pool') is randomly selected from the population (with the pool size set by the user), and the fittest individual among them is selected as the parent [147]. Tournament selection is straightforward to implement and provides intuitive control over selection pressure (how strongly the algorithm favours fitter individuals) by varying the tournament size, making this method one of the most used approaches in GA literature [149].

From these three options, tournament selection was chosen for this project due to its simplicity, robustness and intuitive control over selection pressure. Unlike roulette wheel selection, it performs reliably even when population members have similar fitness scores, and it is not skewed by outliers. Compared to rank-based selection, it avoids the added complexity of defining rank-to-probability mapping schemes whilst still allowing intuitive control over selection pressure via the tournament size. These advantages, combined with

its strong performance and widespread use in literature make tournament selection a well-suited choice for this application.

Crossover Operator

For performing crossover in GAs, the three most commonly used methods are:

Single-point crossover: One crossover point is chosen randomly, and the halves form both parents are swapped to produce the two child solutions [150] (Fig. 3-20).

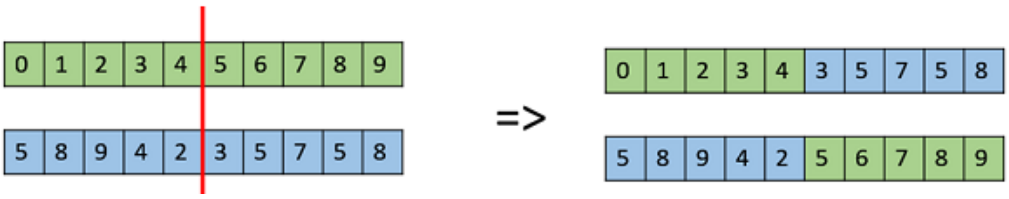


Fig. 3-20. Illustration of single-point crossover. Image from [151].

Multi-point crossover: Two or more crossover points are generated and the segments between points are alternated to form the child solutions [150] (Fig. 3-21).

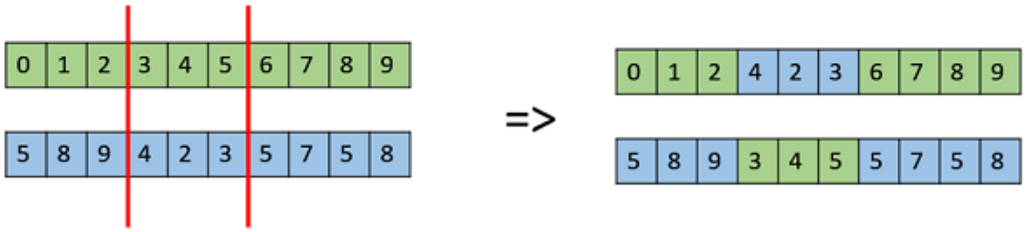


Fig. 3-21. Illustration of multi-point crossover. Image from [151].

Uniform crossover: Each gene in the offspring is chosen independently from one of the two parents with a fixed probability (typically 50%), producing a uniformly mixed combination of both parents' genes [150] (Fig. 3-22).

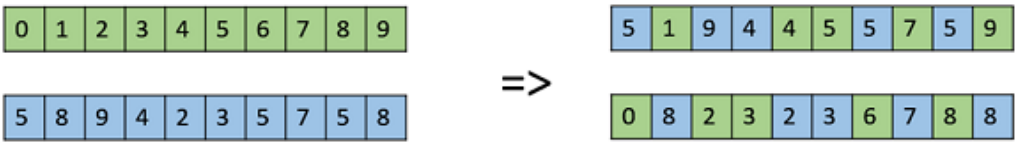


Fig. 3-22. Illustration of uniform crossover. Image from [151].

For this project, single point crossover was selected to ensure minimal disruption to the existing gene structures (i.e. to ensure that both child solutions can inherit 'good parts', such as consecutive genes, from both parent solutions). It is well known in GA literature that crossover methods with high gene disruption (such as multi-point or uniform crossover) often give offspring which are very different from both parent solutions due to the large amount of gene recombination [149], [152-154]. Whilst potentially beneficial for promoting population diversity, this can lead to slower convergence as good local gene structures can easily be lost.

Additionally, rather than using the same crossover point, different random crossover points are used for both parents to allow the child chromosomes to vary in length (Fig. 3-18). Given that it is difficult to know beforehand what an 'ideal' number of cuts would be for a structure, allowing the length of chromosomes to change is beneficial as it introduces a natural way to vary the number of cuts in child solutions.

Mutation Operator

Unlike selection and crossover, which typically rely on established formats (e.g. tournament selection and single point crossover), mutation is a more problem specific operator which is typically tailored based on the problem and how the solutions are encoded. For example, if solutions are encoded as a binary string of ones and zeros, mutation could involve flipping a bit from a 0 to a 1. In permutation-based problems (such as order optimisation in packing), this could involve swapping the position of two objects in the permutation. For real-value encoded problems (such as this cutting and packing algorithm), mutation might add a small amount of random noise to a gene.

In this project, each gene represents a cut in 2D (or 3D) space defined using a set of real values. The mutation operator was designed to apply small, random perturbations to individual cuts, enabling the fine-tuning of their positions without significantly disrupting the overall cutting pattern.

One consideration across all problem types, however, is how aggressively to apply mutation, i.e. whether to mutate all genes in a chromosome or only a small number. Mutating all genes introduces greater diversity but risks erasing beneficial gene structures (i.e. good partial cutting patterns) inherited during crossover. In contrast, applying mutation to only a small number of genes helps preserve good partial gene structures whilst still allowing for some diversity to be introduced. For this reason, mutation in this algorithm is applied on a per-gene basis, with each gene having only a small chance of being mutated. This allows individual cuts to be adjusted whilst other remain unchanged, allowing some

cuts to be refined without excessively disrupting good partial gene structures inherited from crossover.

3.3.4 Importance of GA Parameters

A key aspect of genetic algorithm performance lies in managing the trade-off between exploration and exploitation. Exploration involves searching over a wide and diverse range of solutions to avoid premature convergence to poor ones. Exploitation focuses on the improvement of promising existing solutions to try and improve them further.

Striking the right balance between the two is critical for the success of GAs. Leaning too heavily toward exploitation (i.e. aggressively refining only the best-found solutions) risks premature convergence to poor solutions. In contrast, excessive exploration, whilst helping to promote diversity, can lead to slow and inefficient search behaviour, preventing the algorithm from converging to high quality solutions. A well-tuned GA must therefore balance both aspects: it must maintain enough diversity to explore broadly whilst also focusing search efforts to refine and exploit the most promising solutions found so far.

In this implementation, crossover acts as the primary method of exploitation by recombining genes from well performing parents to produce improved offspring. In contrast, mutation acts as the primary source of exploration by making small random changes to individual genes, helping to promote diversity and explore new regions of the search space. It is worth noting that, whilst crossover can introduce large changes to chromosomes (by varying the number of cuts), it remains fundamentally exploitative in nature since it can only recombine existing genes (but cannot introduce new ones).

This balance between exploration and exploitation is further influenced by several key GA parameters:

- **Tournament Size:** Controls selection pressure. Large tournament sizes favour fitter individuals, increasing exploitation. Smaller sizes increase exploration by allowing less fit individuals to contribute to the gene pool.
- **Mutation Rate:** Higher mutation rates mutate more genes in a chromosome, promoting exploration, but at the risk of disrupting good gene structures. A low mutation rate promotes exploitation, but can lead to premature convergence.
- **Elite Population Size:** Elitism preserves top performing individuals across generations, enhancing exploitation. However, a large elite population reduces the

number of new offspring, potentially limiting exploration. A smaller elite population increases diversity but risks losing high quality solutions.

- **Population size:** A larger population offers greater diversity, improving exploration but with higher computational cost due to more solution evaluations per generation. Conversely, smaller populations reduce computational cost but may lack sufficient diversity.
- **Number of Generations:** More generations allow for longer refinement, improving exploration, but with diminishing returns over time. Beyond a certain point, improvements tend to plateau [149], [155], [156] while computational cost continues to rise.

Together, these parameters must be carefully balanced to ensure the algorithm effectively searches the solution space while steadily improving solution quality over time.

3.3.5 Parameter Selection Strategy

Given the computational complexity of the cutting and packing problem, performing rigorous parameter tuning through extensive testing was not possible. For the 2D work presented in Chapter 4, a more informal approach of trial-and-error was used to select suitable parameter values based on observed performance trends in the algorithm. For the 3D work presented in Chapter 5, the effect of the different parameters is explored in greater detail, with particular focus on mutation rate, tournament size and population size.

Regarding the number of generations, a fixed value of 100 was used for both the 2D and 3D testing of the cutting and packing GA. This value was chosen based on observations in literature and initial testing which found that the majority of performance improvements in the GA tend to occur within the first 0-50 generations, with diminishing returns beyond that point. Using 100 generations was therefore seen as a reasonable trade-off between ensuring that the algorithm has sufficient opportunity to evolve high quality solutions and keeping the computational costs manageable.

3.3.6 Implementation and Software Development Effort

Whilst the previous sections outlined the methodological and algorithmic aspects of the proposed cutting and packing framework, it is also important to clarify the extent of software development required to realise this methodology in practice. A large portion of

the work presented in this thesis involved original algorithm implementation, software integration, and iterative testing and debugging across multiple software environments.

As outlined in Section 3.1.1, the 2D implementation of the cutting and packing framework was developed primarily in MATLAB, with DigiPac used as an external voxel-based packing engine. Enabling this workflow required interfacing MATLAB with the existing C++ DigiPac codebase. The necessary modifications to allow DigiPac to be called from MATLAB was carried out by the primary supervisor of this project (Dr Xiaodong Jia), however all cutting logic, optimisation frameworks (including genetic algorithm operators, cost evaluation criteria and constraint handling policies), and experimental procedures for the 2D work was implemented entirely by the author of this thesis in MATLAB. This included the development of the full 2D cutting and packing prototype, the hyper-heuristic packing framework, the allocation-based packing strategy, and the disassembly sequencing algorithms presented in Chapter 4.

It is also worth noting that the disassembly sequencing work did not rely on DigiPac and was implemented entirely within MATLAB by the author, including the development of the underlying tree-based search logic and constraint checking mechanisms.

For the 3D implementation presented in Chapter 5, the algorithm was integrated into the decommissioning software NuPlant, which is written in C++ and operates entirely in voxel space. The core implementation of the 3D cutting and packing framework within NuPlant was also carried out by Dr Xiaodong Jia, based on the methodology and algorithmic design developed by the author. The author's contribution at this stage focused on defining the algorithm structure (including the GA feedback loop, custom genetic operators and multi-container packing logic) as well as validating the implementation and conducting extensive testing and debugging. This involved the repeated execution of the 3D algorithm across multiple test cases, identification of edge-case failures, and detailed feedback to guide refinements to the implementation.

Across both the 2D and 3D phases of the project, the work required the design, implementation and validation of several novel algorithmic frameworks developed as part of this research, including a 2D prototype of the cutting and packing algorithm, hyper-heuristic and multi-container packing strategies and disassembly sequencing optimisation algorithms. While exact line counts are not reported due to iterative development and refactoring, the overall implementation effort comprised several thousand lines of original MATLAB code and substantial modifications to existing C++ codebases. This section is

intended to clarify the non-trivial software engineering effort underpinning the research contribution presented in this thesis.

CHAPTER 4
2D WORK

This chapter presents the 2D implementation and testing work conducted to evaluate and refine the proposed cutting and packing optimisation approach. The 2D setting provides a simplified environment for testing the methodology and exploring ways to enhance the algorithm's performance and applicability before extending it to full 3D implementation. The findings from this Chapter were instrumental in informing the design choices for the 3D implementation of the algorithm (outlined in Chapter 5).

The remainder of this chapter is structured in four parts:

- **Section 4.1 - Evaluating the proposed approach:** Implementing a simplified 2D version of the algorithm and benchmarking its performance against a random search to evaluate the effectiveness of the proposed optimisation strategy for cutting and packing.
- **Section 4.2 - Hyper-heuristics for container packing**: Implementing and testing a novel hyper-heuristic packing framework for improving both single-container and multi-container packing performance by optimising placement strategies.
- **Section 4.3 - Allocation-based multi-container packing**: Comparing a prior-allocation approach to an order optimisation approach for multi-container packing to assess the difference in terms of speed and solution quality.
- **Section 4.4 - Disassembly sequencing for cut structures**: Developing and testing several optimisation approaches for finding feasible and mechanically stable disassembly sequences for cut structures.

4.1 Evaluating the Proposed Approach

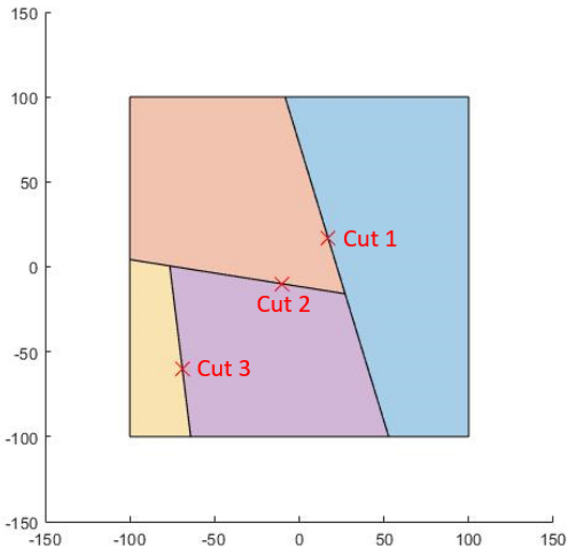
To evaluate the efficacy of the proposed cutting and packing optimisation approach, a simple version of the algorithm was implemented in 2D in MATLAB. The performance of the algorithm was then benchmarked against a random search to establish a clear performance baseline. The random search provides a simple unintelligent approach for exploring the search space, making it an effective point of comparison for demonstrating the benefits of evolutionary principles such as crossover and mutation. By comparing the results to this baseline, it allowed an easy assessment on whether the proposed GA

approach can consistently outperform a non-heuristic approach in terms of quality and convergence speed.

4.1.1 Implementation

As stated in Chapter 3, the chosen cutting approach used to partition the input shape is Binary Space Partitioning (BSP). The implementation of this cutting approach is based on the implementation presented in [109] (Where BSP is used to partition 3D models for additive manufacturing), and is outlined as follows:

Referring to Fig. 4-1 as an example, cutting patterns are represented as a list of successive cuts, where each entry in the list stores the location and orientation of a single planar cut. To apply a cutting pattern to the object, the algorithm applies the cuts one by one, starting with the first cut at the top of the list. For each cut, the origin point of its rotation axis is given by (x, y), and the orientation about this point by θ. The length of each cutting plane is set to be long enough to fully intersect with the object (to prevent partial cuts where the end of a cut stops mid-way through the object), however cuts are not allowed to intersect with one another. To keep the search space computationally tractable, the location and rotation values for each cut are restricted to integer values (otherwise the number of possible cut configurations would be infinite).



(a)

	x	y	θ
Cut 1	17	17	172
Cut 2	-8	-10	97
Cut 3	-67	-59	178

(b)

Fig. 4-1. Simple illustration of Binary Space Partitioning. (a) – BSP cutting of a square with 3 cuts. (b) – cut list used to define the 3 cuts.

Regarding the crossover and mutation operators in the GA, when crossover is performed, the two parent cut lists (selected via tournament selection) are split at random points and recombined to produce the two offspring (Fig. 4-2). The child solutions produced by crossover are then subjected to mutation, where the algorithm will cycle through all the individual cuts in a cut list with each cut having a small probability of being mutated (Fig. 4-3). When a cut is mutated, a small amount of random noise is added to both the location of the origin point (x, y) and the angle θ, to slightly perturb them. Additionally, if the origin point of a cut lies on a previously placed cutting line, or if the origin point lies outside the objects bounding box (both of which can occur due to mutation), the cut will be deleted.

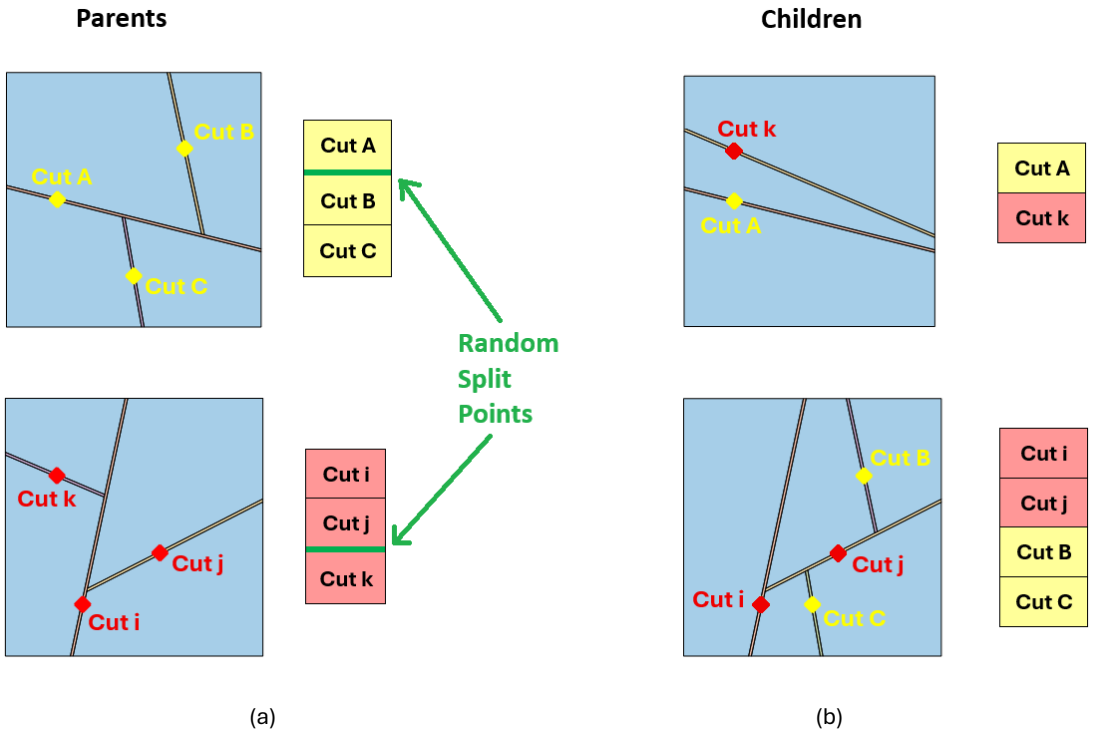


Fig. 4-2. Illustration of crossover. (a) – Parent solutions. (b) – Recombined child solutions.

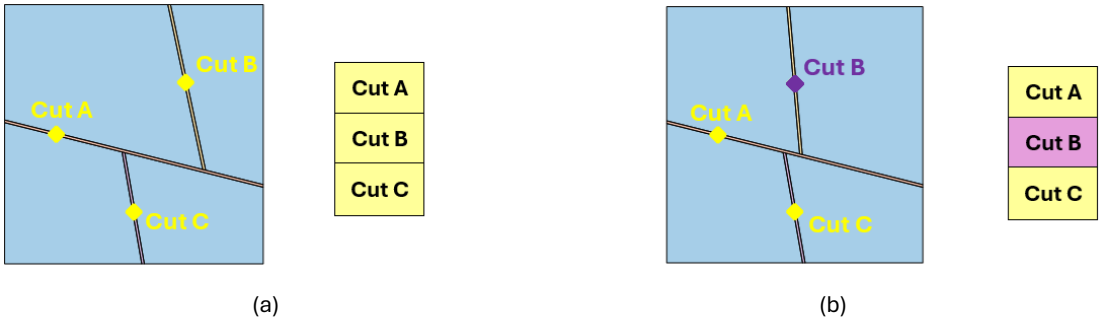


Fig. 4-3. Illustration of mutation. (a) – Unmutated child solution. (b) – Mutated child solution with position and orientation of cut B mutated.

For packing the cut parts, the MATLAB code was interfaced with the packing software DigiPac. The shapes themselves are defined as a set of connected vertex coordinates. To pack a set of shapes, the vertex coordinates for each shape are first written to a text file. MATLAB then calls DigiPac which reads the shapes from the text file, voxelises them and then packs them one by one in the order they appear in the text file. Once packed, DigiPac converts the packed shapes back to vertex coordinates which are then printed to another text file (along with the relevant packing metrics). The text file is then read back into MATLAB to obtain the packing configurations for display.

As stated in Chapter 3, At the time of this study, DigiPac did not have multi-container packing capabilities implemented. As such, a single unbounded container with fixed width and infinite height is used. The packing of the parts is achieved by placing them one by one into the container, with each part being placed according to the placement heuristic 'height minimisation' (which seeks to minimise the height of the object in the packing pile). When searching for valid packing sites for a part, the parts are allowed to fully rotate in 1-degree increments. For the packing order of parts, the parts are packed in the order they are produced from the cutting algorithm. When the cutting algorithm applies a cutting list to the object, cut parts will be appended to an array in the order they are cut from the structure, it is this order in which the objects are placed into the packing space.

The cutting cost is defined as the intersection volume between the cuts and the structure, and the packing cost is defined as the percentage space utilisation of the container. Since the goal would naturally be to maximise the percentage utilisation, the value is multiplied by -1 to convert it to a minimisation problem. This was done for practical reasons, since the MATLAB function used for finding the pareto frontier assumes that all input objectives are to be minimised.

The problem is solved in a multi-objective fashion, where the two objectives to minimise are the 'cutting cost' (intersection volume from cutting) and the 'packing cost' (negative percentage space utilisation). Regarding constraints, the user must set maximum x and y size limits for the cut parts (where the limits must be less than or equal to the container dimensions to ensure the parts can be packed). If either the x or y dimension of a cut part's bounding box exceeds these limits, the solution containing the part will be deemed infeasible, and a penalty term will be used to degrade *both* the cutting cost and packing cost. This penalty is calculated individually for each solution in the GA population during fitness evaluation to ensure that infeasible solutions are appropriately penalised during selection.

This dual penalisation of both objectives is necessary to ensure that the multi-objective search remains consistent across both objectives. For example, if only one objective is penalised, the algorithm may mistakenly favour infeasible solutions that appear optimal for the unpenalised objective. By degrading both the cutting and packing costs, the algorithm is better able to differentiate between feasible and infeasible solutions within both objectives. It is worth noting that both penalty terms are tied specifically to the part size constraint (i.e. if the part size constraint is violated, the penalty terms for both cutting and packing will increase in proportion to the degree of violation).

For cutting cost, the penalty is calculated based on how much the parts violate the size constraint (Fig. 4-4). The algorithm will cycle through all the cut parts in a solution and check each part's bounding box dimensions against the limits. Referring to Fig. 4-4 as an example, for each cut part which is larger than the limits, the algorithm will calculate how much larger the part's bounding box is (both the x and y dimension), compared to the user defined x and y limits. The differences in both dimensions (Δx and Δy in Fig. 4-4) are summed to get the penalty value for that single part. Once the algorithm has cycled through all the cut parts in a solution, the penalty value for each infeasible part is summed and then multiplied by the penalty multiplier, before being added to the cutting cost for that solution.

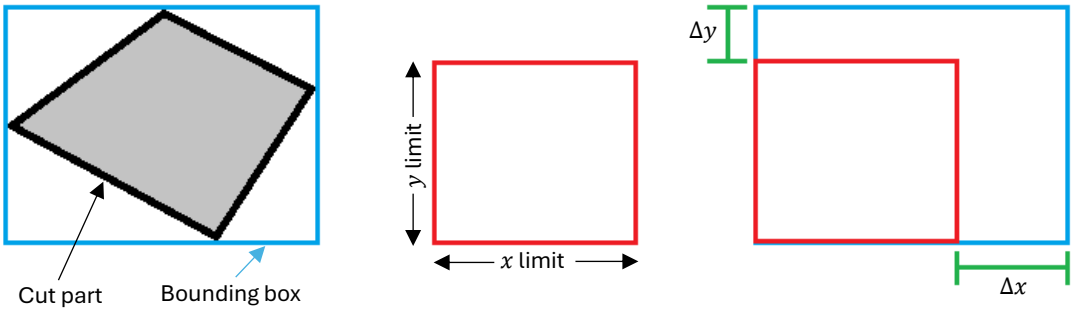


Fig. 4-4. Penalty calculation for a cut part. (a) – Cut part with bounding box. (b) – Maximum allowed size for cut part. (c) – Size difference in x and y dimension.

For packing, only parts smaller than the size limits are packed, and the penalty term is calculated based on how many of the parts violate the constraint (i.e. how many parts are unpackable). For each infeasible solution, the total volume of all the unpacked parts is divided by the total volume of all the parts in that solution. This gives a value between [0,1], equal to 1 if no parts are packed and 0 if all parts are packed. As with the cutting penalty,

this value is multiplied by a penalty multiplier before being added to the packing cost of that solution.

As stated in Chapter 3, for ranking the solutions, the population is sorted into successive pareto fronts (a common approach in evolutionary multi-objective optimisation [157]). The process works by calculating the Pareto front for the entire population (i.e., the first frontier); the solutions on this frontier are then removed, and the algorithm computes the next frontier from the remaining solutions. This process repeats until all solutions have been assigned to a front. This results in a population ranked by dominance level, where solutions in earlier (lower-numbered) frontiers are considered superior to those in subsequent ones.

For crossover and mutation, parent solutions are selected via tournament selection, where a random subset of the population (the tournament pool) is selected, and the best solution amongst them is chosen as the tournament winner. Since solutions are ranked by pareto front, solutions in earlier fronts dominate those in later ones. However, solutions within the same frontier are considered non-dominated with respect to each other, meaning they are treated as equal in terms of fitness. As such, if multiple solutions in the tournament pool belong to the highest-ranked (non-dominated) pareto front, one is selected at random among them. This ensures that all equally ranked top-performing solutions have a fair chance of being selected, helping to maintain diversity within the population.

4.1.2 Simulation Setup

For testing the algorithm, a simple scenario was devised where the goal was to cut a square (with length and width equal to 200 pixels) and then pack the parts into a container (with a width of 200 pixels) such that both the cutting cost and packing cost are minimised. Both the maximum allowed x and y size limits were set to 100. In doing so, it is plain to see that the best possible solution for this problem (i.e. the global optimum) is when the square is cut into 4 equal parts which are then packed to reconstruct the original input square in the packing space (Fig. 4-5 below).

(a) (b)

Fig. 4-5. (a) – globally optimal cutting result. (b) – globally optimal packing result.

To assess the performance of the GA, a popular metric often used in pareto-based multi-objective optimisation, called the 'hypervolume' indicator [158], is used (Fig. 4-6). The hypervolume is calculated by forming a polytope between a reference point in the objective space and each pareto optimal solution and then calculating the volume of the union of all the polytopes [159]. The objective space refers to the space defined by the objective functions being optimised (e.g. cutting cost and packing cost), where each solution can be represented as a point within this space, based on its objective values. This measure was first referred to as the 'size of the space covered' in [160] and essentially shows how the pareto optimal set of solutions improves over time. A higher hypervolume indicates better performance, as it means the set of solutions dominates a larger region of the objective space (reflecting both improved quality and diversity in the trade-off solutions). Fig. 4-6 illustrates this concept for the dual objective cutting and packing problem.

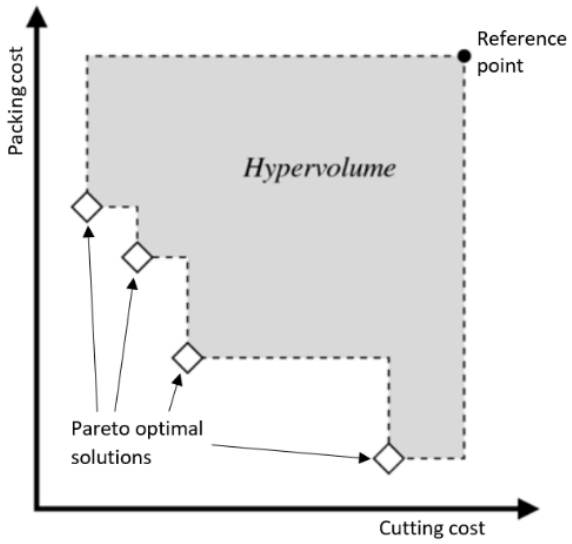


Fig. 4-6. Illustration of the hypervolume in 2D objective space. Image adapted from [161].

For the GA, the following parameters were used:

- **Population size:** 50
- **Generations:** 100
- **Elite population:** 10 (20% of the population)
- **Tournament size:** 5 (10% of the population)
- **Mutation probability:** 0.2
- **Cutting and packing penalty weights:** 100

The GA parameter values outlined above were selected in accordance with the general discussion on GA parameter effects and the parameter selection strategy outlined in Chapter 3 (Sections 3.3.4 and 3.3.5). Given the computational expense of evaluating each cutting and packing solution, exhaustive parameter tuning was not feasible. Instead, parameters were chosen through informal trial-and-error based on observed performance trends (following standard GA design practice which aims to balance exploration and exploitation). In particular, the mutation probability, tournament size, population size and number of generations were chosen to maintain sufficient population diversity whilst ensuring convergence within manageable computational limits.

The initial population was generated randomly, with both the number of cuts in a solution and the location/orientation of each cut generated at random. Elitism was applied to retain the best 10 solutions across generations, with the remainder of the population generated via crossover and mutation.

The performance of the GA was benchmarked against a random search where 5000 cutting patterns (equal to the total number of solutions evaluated by the GA) were randomly generated in batches of 50 (analogous to the generations in the GA). For both the GA and the random search, the hypervolume was calculated using all solutions evaluated up to that point, not just using the 50 solutions in the current generation/batch. This results in a hypervolume value which can only increase as the pareto frontier of all found solutions improves.

4.1.3 Results

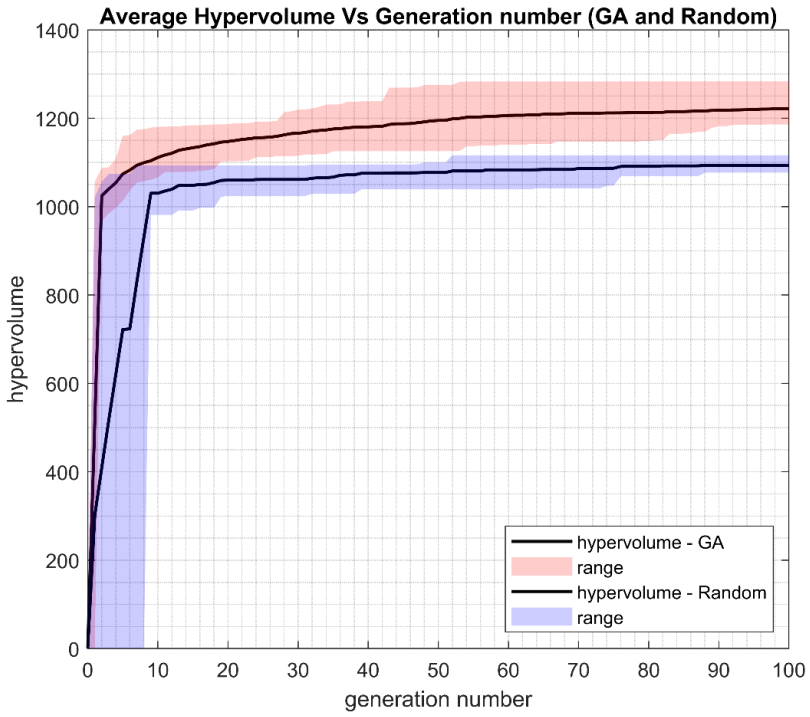


Fig. 4-7. Average hypervolumes for both the GA and random trials.

Both the GA and the random search were run 10 times. Fig. 4-7 shows the hypervolumes for the GA and the random search (plotted against their respective generation/batch number), with the black lines indicating the average over the 10 runs and the shaded regions showing the min/max range. It is immediately evident from this figure that the GA has outperformed the random search, with the final hypervolume values for all the GA runs being higher than that of the random search runs. This shows that the GA can find better approximations of the pareto frontier compared to the random search.

The clear gap between the shaded regions for the two algorithms also shows that in all runs, the GA produces final solutions which dominate all solutions from the random search in the objective space. This is also evident from the example in Fig. 4-8, which shows the two optimised solutions (from all 10 GA and random search runs) that are closest to the global optimum (shortest Euclidean distance to the globally optimum cutting/packing costs in the objective space). It can be seen that in both solutions, the packing structures are similar in terms of percentage space utilisation (with values of 80.1% and 78.4% for the GA and random search respectively), however the cutting solution from the GA has noticeably fewer cuts (with the cutting costs being 92.2 and 152.7 for the GA and random search

respectively). This was a pattern which was observed consistently when examining the pareto optimal solutions produced by the two algorithms over the 10 runs.

The sharp increase in hypervolume seen approximately between generations 0-10 in Fig. 4-7 can be attributed to the use of penalty functions with relatively large penalty multipliers. In the early stages of the search, the majority of candidate solutions are infeasible with respect to the cut-part size constraints, resulting in large penalty values that dominate the objective functions (pushing the solutions far from the feasible region of the objective space). As the optimisation progresses, the population rapidly discovers feasible solutions, at which point, the penalty terms for these solutions are reduced to 0. This transition produces a large discontinuous improvement in the pareto optimal solutions in the objective-space, leading to a pronounced jump in the hypervolume.

This interpretation is further supported by the trends observed in Fig. 4-9 and Fig. 4-10, which show a rapid increase in the average number of cuts in the population and a corresponding drop in the penalty values over the same generation range. Once most of the population lies within the feasible region, further improvements in hypervolume arise from incremental improvements in the trade-off between cutting and packing, resulting in the observed plateau behaviour. Similar ‘cliff-edge’ convergence patterns are commonly observed in penalty based evolutionary optimisation, where early search effort is mainly focused on pushing the search toward feasibility before the objective function itself is improved [162-164].

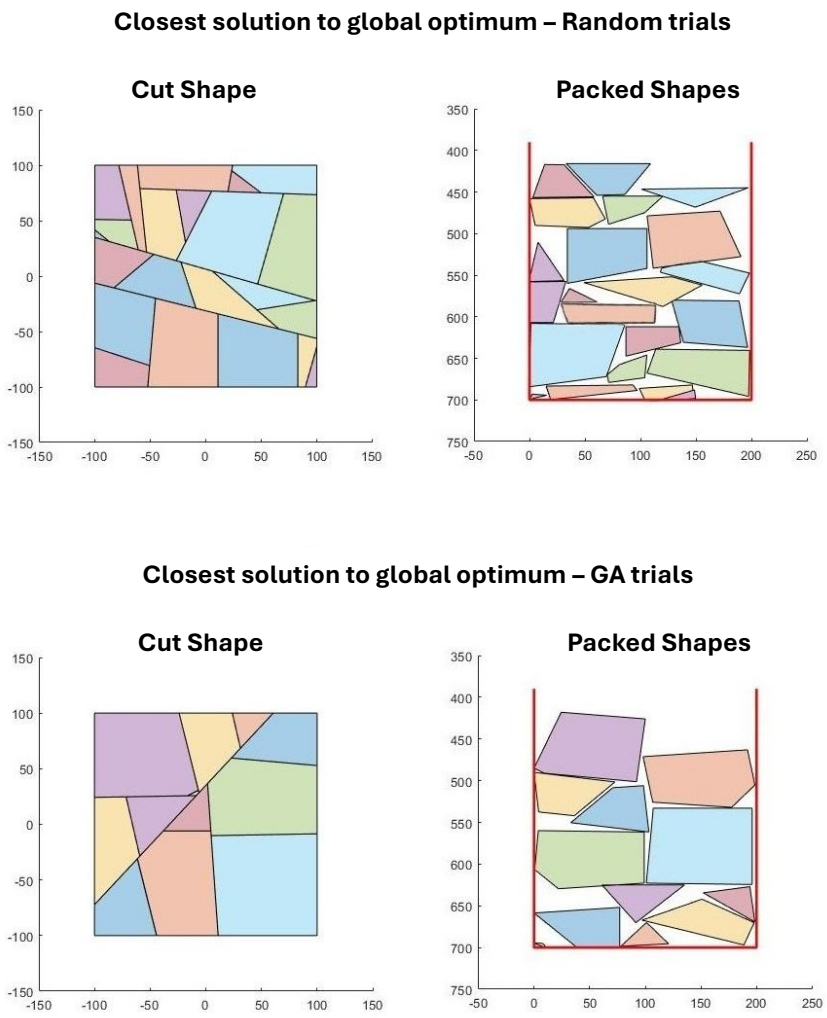


Fig. 4-8. Closest cutting/packing solutions to the global optimum for the GA and random trials.

Based on the results from this study, several problems with the GA were also noted, namely a slow convergence rate (leading to an inability to find the global optimum) and a loss of diversity in the elite solutions.

Regarding slow convergence rate, from the example in Fig. 4-8, despite an average run time of approximately 15 hours, the GA is still unable to produce solutions which are even close to the global optimum (Fig. 4-5). There are several factors which are believed contribute to this slow behaviour:

Number of variables: with the 2D implementation, the number of variables required to represent a single cut is three (x, y, θ). In 3D, this would increase to six $(x, y, z, \theta_x, \theta_y, \theta_z)$. It is well known in optimisation that increasing the number of variables expands the search space, making it more challenging to find optimum solutions. Although three variables per cut may seem modest, this scales quickly with the number of cuts in a solution. Using the proposed BSP cutting approach, the total number of optimisation variables becomes $3n$ for

2D and $6n$ for 3D, where n is the number of cuts. As an example, the closest solution to the global optimum found by the GA (Fig. 4-8) contains 11 cuts, resulting in 33 variables for this cutting pattern. Given that this is a very simple example (i.e. simpler than cutting a real-world structure which could require more cuts), this large number of variables poses a problem in terms of convergence speed (especially if the current approach is adapted to 3D).

Variable ranges: coexisting with the problem of many optimisation variables is the problem of large variable ranges. For each cutting plane, the x and y points that define the origin can take any integer value inside the structures bounding box. Additionally, the cutting planes were allowed to fully rotate about their origin point in 1-degree increments, giving 180 possible planar orientations at any single point in space. For the simple square test scenario, this gives $200 * 200 = 40{,}000$ possible locations for a single plane origin point, with 180 possible orientations at each point, giving the total number of unique positions for a single cutting plane as $40{,}000 * 180 = 7.2 * 10^6$. By cutting down the number of unique positions which individual cuts can take, it follows that the size of the search space will be reduced as well. For example, if the plane rotation increment was restricted to 90 degrees (orthogonal cutting), the algorithm would lose some flexibility in its ability to place cuts, but the number of unique positions would only be $40{,}000 * 2 = 80{,}000$.

Use of penalty functions: The problem with using penalty functions to penalise constraint violation is that it allows infeasible solutions in the GA's population. Whilst this can be beneficial in promoting better exploration of the search space, it also means some of the computational effort will be wasted evaluating infeasible solutions (which can slow convergence). Another problem is the sensitivity to the choice of penalty multiplier. If the penalty term is too low, the algorithm may tolerate infeasible solutions for too long, leading to slow convergence to a feasible optimum, or even a failure to find a feasible optimum within the allocated time. If the multiplier is too high the fitness function becomes dominated by the penalty term, leading to aggressive enforcement of feasibility at the expense of exploration of the search space.

In this study, a high penalty multiplier was used to strongly discourage constraint violations with respect to the part size limits. As noted earlier, this approach succeeds in quickly pushing the population toward feasibility early in the search, however it does so by rapidly increasing the number of cuts (since adding cuts is an easy way to reduce the size of parts). This behaviour is evident in Fig. 4-9 and 4-10, which shows a sharp initial rise in the average number of cuts in the population accompanied by a corresponding drop in the average population penalty values. The problem however is that once the population reaches the

feasibility region, many of the population members contain more cuts than is necessary. Because removing cuts risks easily introducing constraint violations, the algorithm proceeds slowly, making only minor improvements to the existing solutions. As such, it becomes very difficult for the algorithm to quickly find feasible solutions with few cuts, leading to slow convergence.

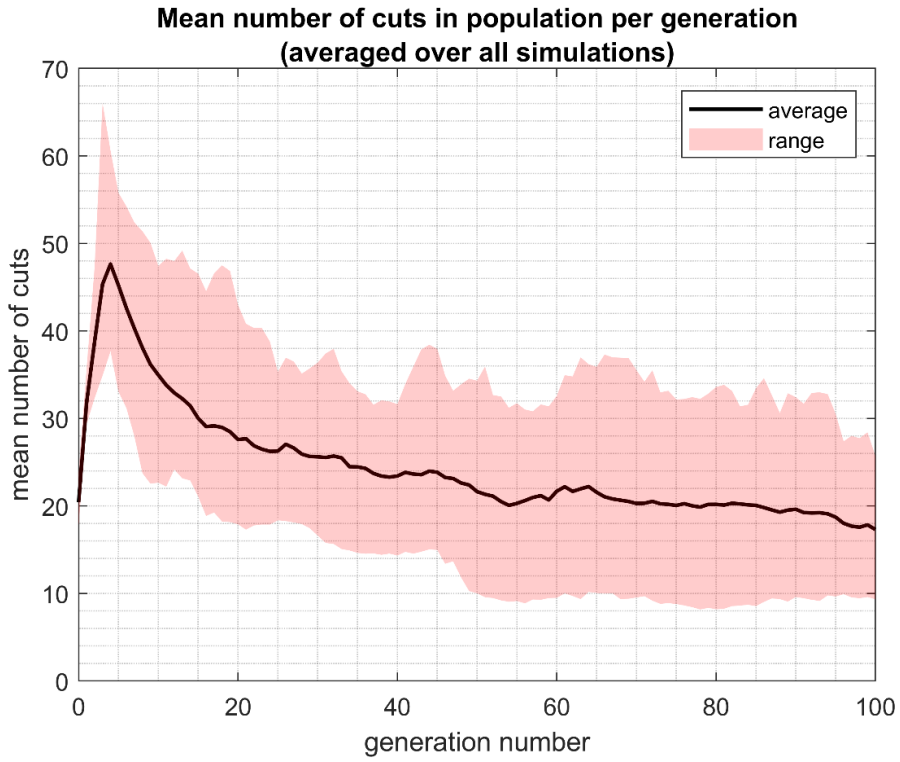


Fig. 4-9. Mean number of cuts in the population per generation (averaged over all simulations).

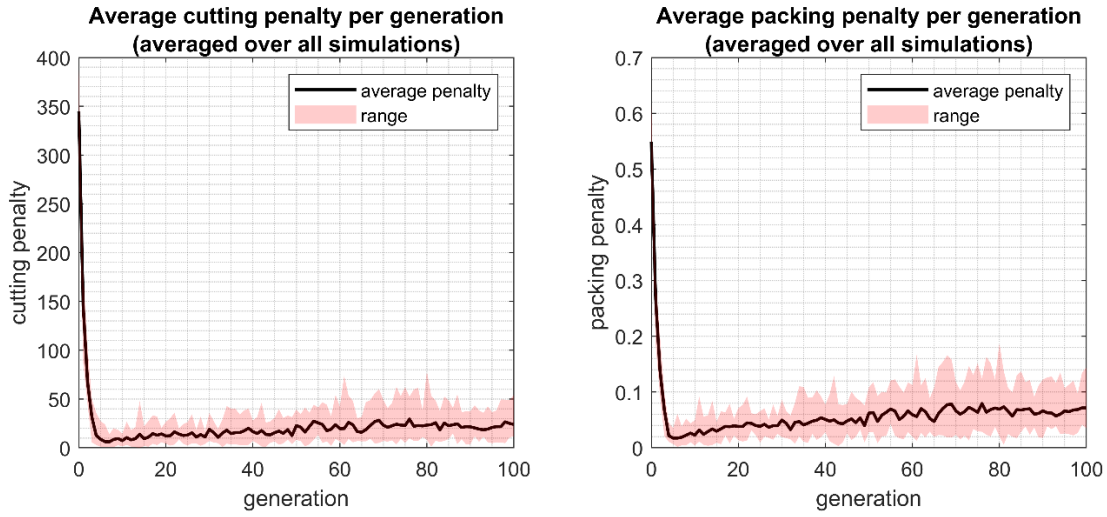


Fig. 4-10. Mean raw cutting and packing penalty for the population, per generation (averaged over all simulations).

Regarding a loss in diversity, upon closer inspection it was also noted that many of the elite population members in the final generation of each GA run were very similar. As an example, Fig. 4-11 shows two elite solutions from the final generation of the best GA run. It is clear that both cutting patterns appear very similar, with only minor differences between them. It is also clear that, whilst the packing structures are visually different, they both have similar peak heights, leading to similar packing costs. This was a pattern which was observed across all the elite solutions in the final populations of the 10 GA runs.

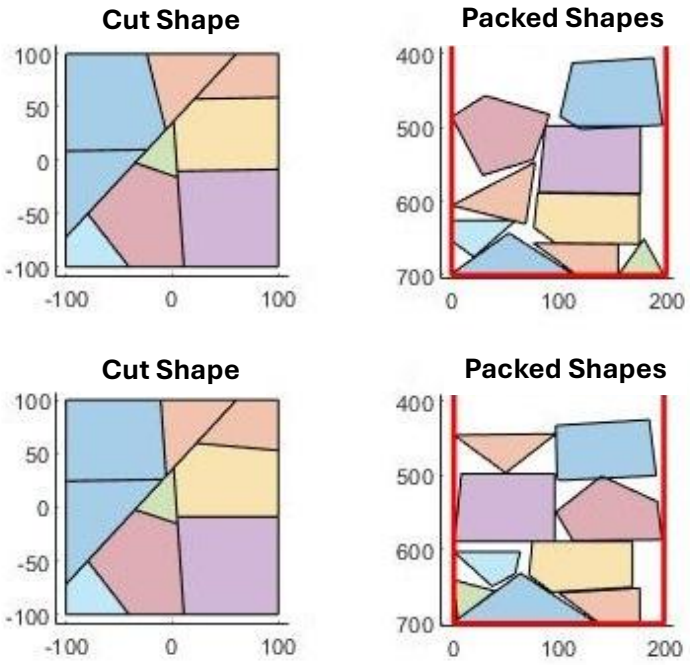


Fig. 4-11. Example of diversity loss in cutting patterns within the elite population.

This issue is likely due to the choice of large elite population size. Given the complexity of the problem and the low likelihood of obtaining globally optimal solutions in realistic scenarios, the initial design of the algorithm emphasised maintaining a strong set of high-quality solutions throughout the run. A large elite population size was intended to preserve the top-performing solutions across the generations, allowing the algorithm to incrementally refine them. The problem however is that in certain cases, the genetic operators would produce child solutions which were only marginally different from their parents. When this occurred within the elite subset, these similar offspring were often added back into the elite population alongside their parents, further reducing diversity. In retrospect, a smaller elite population would have likely struck a better balance between preserving quality and maintaining the exploratory capacity of the algorithm.

4.1.4 Concluding Remarks: Evaluating the Proposed Approach

Based on the study presented, the efficacy of the proposed optimisation methodology was verified by showing that it can consistently outperform a brute force random search. Based on the results, several problems were noted with the current implementation of the GA, namely:

1. **Slow convergence:** The combination of penalty functions coupled with the high number of optimisation variables (due to the representation of cuts) likely increases the time needed to find good feasible solutions. To help mitigate this issue, the fallback cutting strategy (orthogonal cutting) and repair mechanism (presented in Chapter 3, Section 3.1.2 and 3.1.6, respectively) will be implemented in the 3D algorithm, thereby reducing the number of optimisation variables and removing the need for penalty functions.
2. **Loss of diversity in the elite population:** A large elite population increases the chance that elite members are selected for crossover and mutation. Since these operators can produce offspring which are genetically similar to their parents, this can lead to a concentration of similar solutions in the elite population. To help mitigate this issue, a smaller elite population size will be used for the 3D implementation of the algorithm.
3. **Lack of multi container packing:** The current implementation of DigiPac does not have multi container packing optimisation implemented. Given that nuclear decommissioning often involves large structures which need to be packed into multiple industry standard containers, multi container packing remains a high priority direction of research.

4.2 Hyper-Heuristics for Container Packing

From the review on 3D irregular object packing optimisation, it was noted that the number of studies investigating hyper-heuristics in 3D packing is very limited (with the only example being [31]). As a reminder to the reader, within the context of packing optimisation, ‘hyper-heuristics’ refers to a high-level selection strategy designed to select low-level heuristics for solving the packing problem. As an example, this could involve using a set of different placement heuristics (i.e. rules to decide where objects are placed in the container, like height minimisation) which the algorithm can select from, rather than using only a single placement heuristic. In doing so, the algorithm gains more flexibility in its choice of placement locations when packing objects, potentially allowing it to find better solutions.

Given the limited research on this topic, and the fact that DigiPac is a one-by-one packing algorithm which utilises placement heuristics, investigating hyper-heuristics for improving the quality of packing solutions presented itself as a natural choice of research direction.

The following two sections present an investigation into the use of hyper-heuristics for 2D single-container and multi-container packing (Sections 4.2.2 and 4.2.3 respectively). For the single container case, an optimisation algorithm is proposed with 4 placement heuristics combined in a novel hyper-heuristic fashion, where objects are packed in a fixed packing order (largest volume to smallest), with a GA used to optimise the choice of placement heuristic for each object. The proposed framework is then compared to a conventional single-heuristic order-optimisation approach to assess whether the proposed hyper heuristic framework can outperform it. In the multi-container work, the hyper-heuristic framework is extended to multi-container packing by combining heuristic optimisation with order optimisation, where the goal of the GA is to optimise both the packing order (which changes the allocation of objects to different containers), and the placement heuristic used to pack each object (with the aim being to minimise the number of containers required to pack all parts). The proposed algorithm is then benchmarked against literature, and based on the findings, and updated hyper-heuristic framework is proposed.

4.2.1 Background

Whilst hyper heuristics have received almost no attention in 3D packing, the number of papers focusing on hyper-heuristics for irregular object packing in 2D is also very limited, with the majority of work being from the same research group [165-170].

In [165], the authors introduce a first-version of their hyper-heuristic framework which uses an adaptive rule-based system to evolve combinations of low-level heuristics (using a genetic algorithm) to solve both regular and irregular 2D packing problems. In particular, they use two types of low-level heuristics for solving the problem: one set of heuristics which are designed to select which object to pack next and which container to place it into (e.g. first fit decreasing where objects are ordered from largest volume to smallest and the largest piece is placed into the first container that will accommodate it), and another set of heuristics for deciding where to place each object in the container (e.g. bottom left fill where the object is placed as low as possible in the container). The decisions are based on measurable properties of the current packing situation (referred to as the ‘problem state’),

such as how many pieces are left to pack, the size and shape categories of those pieces, and how much of a container is already filled.

In [166], the authors apply their hyper-heuristic framework in a multi-objective fashion for solving 2D irregular stock cutting problems, where the two optimisation objectives are to minimise the number of containers for all the parts and the time required to place all the objects. In the same year, in [167], the authors build on their proposed hyper-heuristic framework by utilising data mining techniques to automatically define a set of problem features (rather than having them defined manually as before). The features help the hyper-heuristic make better decisions by identifying patterns in the heuristic performance across different scenarios, helping to improve the selection of heuristics. In [168], the authors extend their framework to handle both 1D and 2D packing problems, showing that the methodology can generalise well to different types of packing problem. This idea was further consolidated in [169], where the authors propose a unified hyper-heuristic framework for solving 1D, 2D regular and 2D irregular packing problems. They expanded their heuristic set and validate the framework on 1400 benchmark instances, demonstrating its adaptability and robustness for a wide range of problems. Finally, in [170], they extend their hyper heuristic framework to a multi-objective domain where the two objectives are to minimise the number of containers and the time required to pack all objects. Whilst similar to their earlier work in [166], this version of the algorithm benefits from advances they made in their later work in automatic feature extraction and framework generalisation.

Another example of hyper-heuristics for irregular object 2D packing (from a different research group) can be found in [171]. In this paper, the authors propose a hierarchical hyper-heuristic which shares some similarities with the earlier work by the previous research group. Both approaches use selection heuristics for selecting which object to pack next and placement heuristics to decide where to place each object. The main difference with their work however is that the heuristics operate on one container at a time, trying to fill the container as much as possible before moving onto the next one. Their hierarchical system builds solutions using three types of heuristic operators: selection, placement and hole filling. The selection heuristic selects which object to pack next (based on shape features such as aspect ratio or area) and the placement heuristics determine the placement strategy for the object (e.g. positioning objects from the bottom of the container to the top to form columns or positioning the objects from left to right to form rows). The hole filling heuristic is used at the end to try and fill any remaining gaps in the structure in a greedy fashion.

While the work by the authors of [165-170] and [171] centres on adaptive heuristic selection mechanisms based on problem state features, the approach proposed in this project differs in its use of a global optimisation framework. In their implementations, the hyper-heuristic framework operates as a local decision-making process, i.e. a high-level selection strategy chooses a heuristic at each step based on features of the current packing state. While this enables adaptability, it does not involve systematically trialling a wide range of heuristic combinations, meaning that potentially high-performing combinations may never be explored. For the hyper-heuristic framework proposed in this project, rather than selecting heuristics dynamically using a high-level selection strategy, a genetic algorithm is used to trial a wide range of different heuristic combinations (in the single container packing case), and different heuristic/packing order combinations (in the multi-container case) to identify the combinations which yield the best results. This transforms the hyper-heuristic from a reactive decision process to a more exploratory, global search process offering broader coverage of the search space with the potential to uncover higher quality solutions through iterative refinement.

The final example covered here is presented in [172]. In this paper, the authors propose a hyper-heuristic algorithm for solving the strip packing problem (packing shapes into a single unbounded container). Conceptually, their approach functions similarly to the hyper-heuristic multi-container algorithm developed in this project, in that a genetic algorithm is used to optimise both the packing order of the objects and the selection of placement heuristics. However, there are several key differences. Firstly, their work is limited to single-container packing and does not extend to multi-container scenarios, which is a central focus of the present work. Additionally, their algorithm also optimises the rotation angles for objects within the GA (whereas in the current work, the angles are determined implicitly by the placement heuristics) as well as the granularity of the discretised packing space (in contrast to this study where the grid size remains fixed throughout), increasing the complexity of the search process.

The following sections present the implementation and results of the proposed hyper-heuristic framework, beginning with the single-container version before extending to the multi-container case.

4.2.2 Single Container Packing

To investigate the efficacy of hyper-heuristics, this research began with an MEng project [173] which focused on testing single-container hyper-heuristic packing. The goal of this

project was to implement a simple hyper-heuristic packing algorithm for packing shapes into a single unbounded container (with fixed width and infinite height), and to then compare the efficacy of the hyper-heuristic algorithm to a more conventional single heuristic order-optimisation approach. The project was conceptualised by the author of this thesis, with the programming work also conducted by the author. The testing, data collection and analysis was conducted by the student under close supervision by the author.

The shape dataset chosen (by the student) for testing is referred to as 'Jakobs1' and is available from [174]. Fig.4-12 shows an example of a packed structure using the Jakobs1 dataset.

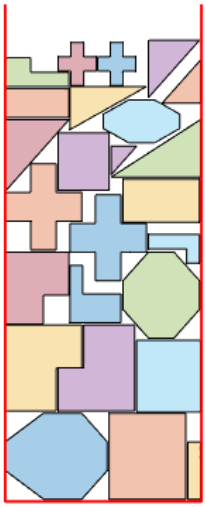


Fig. 4-12. Example of a packed structure created using the Jakobs1 dataset. Image from [173].

For the hyper heuristic algorithm, a placement-heuristic optimisation approach was used where the algorithm has several placement heuristics available which it can use to place objects. The objects were packed one-by-one from largest volume to smallest, each time being placed according to the placement heuristic assigned to it. The goal of hyper heuristic optimisation then, was to optimise the choice of placement heuristics for each object to maximise the percentage space utilisation of the container.

The four placement heuristics used in this study were:

1. **Height Minimisation** – Object placed in a location which minimises its height in the packing pile (Fig. 4-13 a).
2. **Seat Minimisation** – Object is placed in the lowest available location in the packing structure (Fig. 4-13 b). A point to note, this placement heuristic is equivalent to the popular placement heuristic often used in packing referred to as 'bottom left fill' [80].

3. **Contact Area Maximisation** – Object is placed in a location which maximises the contact area between itself and the objects in the packed structure (Fig. 4-14 a).
4. **Contact Number Maximisation** – Object is placed in a location which maximises the number of objects in the packed pile that it is in contact with (Fig. 4-14 b).

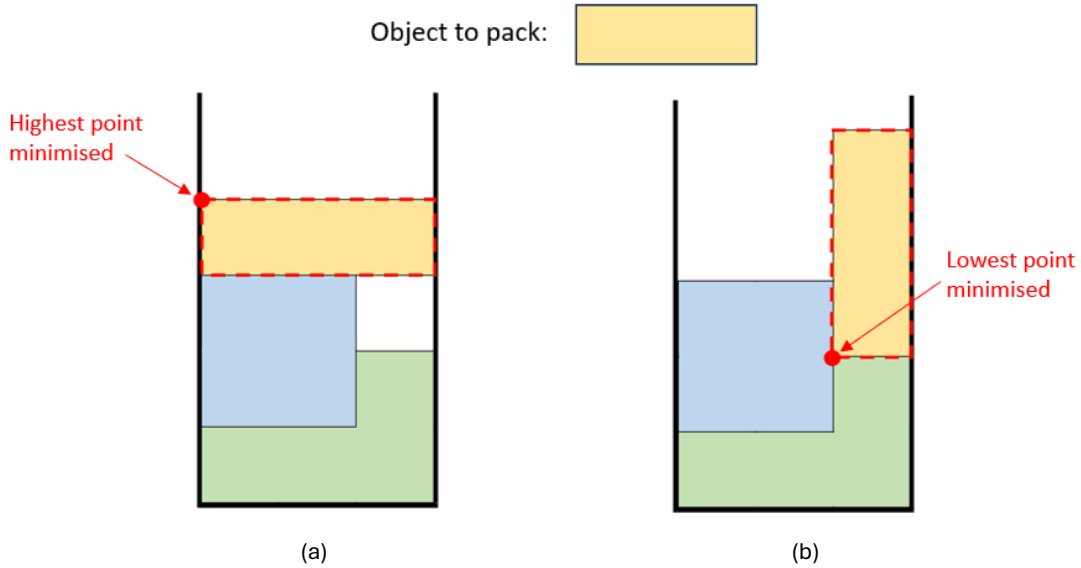


Fig. 4-13. Simple example to illustrate difference between height and seat minimisation. (a) – object placed using to height minimisation. (b) – object placed using seat minimisation.

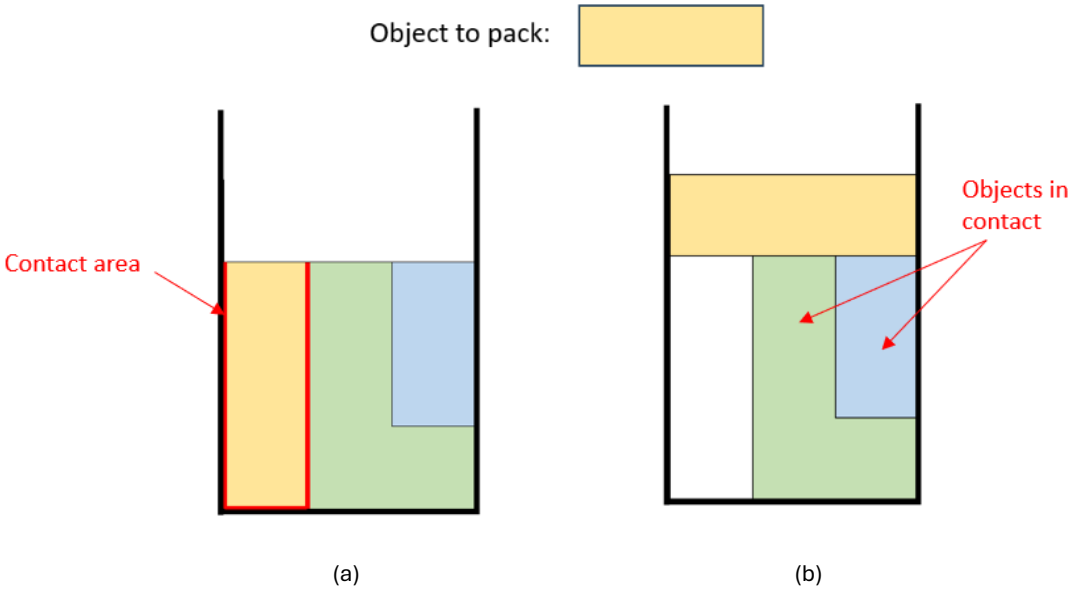


Fig. 4-14. Simple example to illustrate difference between contact area and contact number maximisation. (a) – object placed using to contact area maximisation. (b) – object placed using contact number maximisation.

The motivation for selecting these four placement heuristics was as follows:

- **Height Minimisation** – Based on the literature review on 3D irregular object packing algorithms (Chapter 2, Section 2.2), it was noted that height minimisation is a frequently used choice of placement heuristic. This is likely due to the fact that height minimisation naturally seeks to always keep the height of the packing structure as low as possible, leading to compact solutions with a lot of free space on top of the pile where new objects can easily be added.
- **Seat Minimisation** – Based on the literature review, 'bottom-left-fill' was also noted as a commonly used placement strategy. Bottom-left-fill (which functions the same as seat minimisation) seeks to place objects such that the base of the object is placed as low as possible in the packing structure, with ties being broken by selecting the 'left-most' of the tied positions. The idea behind this placement heuristic is that by placing objects as low as possible in the packing structure, the algorithm fills the container space from the bottom up, helping to minimise large vertical gaps at the bottom of the container. The downside however is that it does not explicitly seek to minimise the height of the packing pile and as such, can produce solutions with objects sticking out of the top of the pile (e.g. Fig. 4-13 b).
- **Contact Area Maximisation** – Contact area maximisation is a less widely explored placement heuristic that is considered more problem-specific when compared to height and seat minimisation. The idea behind contact area maximisation is that it is more beneficial for shapes that have large flat surfaces which can be placed in contact with one another to minimise gaps.
- **Contact Number Maximisation** – Like contact area maximisation, contact number maximisation is also a more problem-specific placement heuristic. Whilst contact area is more beneficial for shapes with large flat sides, contact number maximisation is more beneficial for irregular, curved shapes where maximising contact area may be difficult. As a simple example, consider the packing of circles into a container. Due to the curved nature of circles, it is difficult to maximise contact area between them since the contact area itself is inherently limited. In such cases, maximising the number of other circles in contact will likely lead to better solutions with minimal gaps between them.

For the hyper-heuristic algorithm, a genetic algorithm was used to optimise the choice of placement heuristics. Each chromosome in the GA is a 1-dimensional array where each entry (gene) corresponds to a placement heuristic (represented as a number between 0-3). The entry position in the array (from left to right) corresponds to which object is packed by the heuristic. E.g. the heuristic in position 1 of the array is used to pack the first object, the

heuristic in position 2 the second object, and so on. When performing crossover, single point crossover was used where the algorithm selects two chromosomes, splits them at random points and then recombines the top and bottom halves to produce two child solutions (Fig. 4-15). As a point to note, unlike in the GA used to optimise the cutting patterns in the previous section, the same random split point is used for both parents to ensure that both child solutions remain the same length (since the number of entries in the array corresponds to the number of objects to pack, which remains constant). For mutation, the algorithm takes a single chromosome and randomly changes one of the genes (placement heuristics) for a single object.

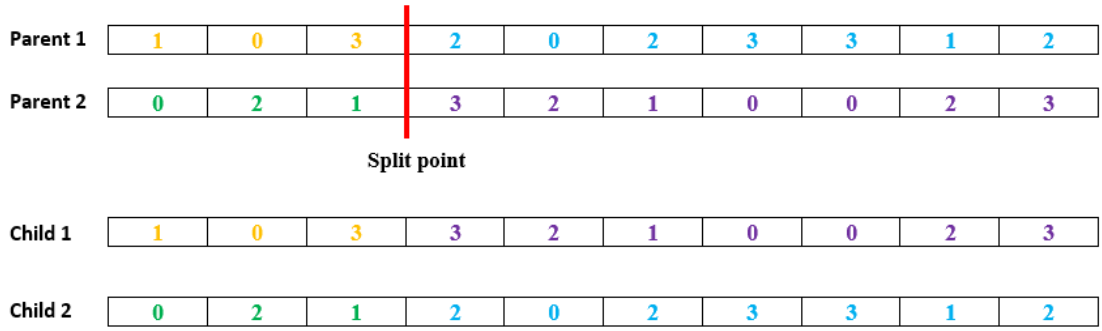


Fig. 4-15. Illustration of single point crossover for a packing case with 10 objects.

For the single heuristic packing, a simple order-optimisation based genetic algorithm was implemented, where each object is packed using a single placement heuristic and the GA optimises the packing order for the objects. Each chromosome in this GA represents a packing order with each gene representing an object. The objects are packed one by one starting with the first entry in the chromosome.

Since the order optimisation problem is a permutation-based problem (where the goal is to optimise the permutation of objects in a chromosome), conventional crossover operators (e.g. random single point crossover) cannot be used as this runs the risk of producing solutions with duplicate or missing objects. As such, a popular crossover operator often used in permutation-based optimisation problems, referred to as Davis's order crossover [175], is used. Referring to Fig. 4-16 as an example, the process begins by copying a random segment of genes (called the 'crossover section') from the first parent chromosome and then placing it into the first child chromosome. Then, starting from the second crossover point in the second parent chromosome, it copies the remaining unused genes into the first child chromosome in the order they appear in the second parent, wrapping around at the

end of the list [175]. The second child chromosome is created using the same process, but with the parent roles reversed.

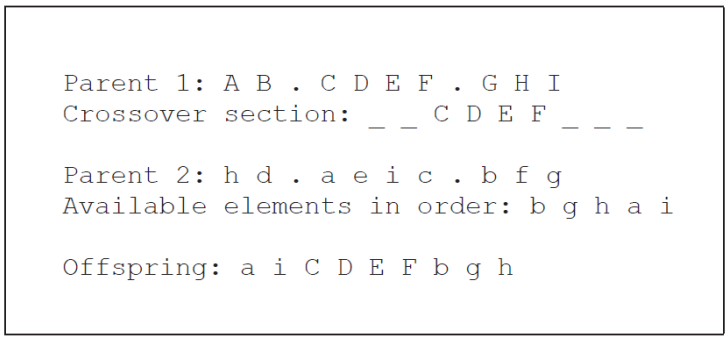


Fig. 4-16. Illustration of Davis's order crossover. Image from [175].

This operator changes the permutation of objects whilst preserving the original object set (i.e. preventing duplicate or missing objects). It also has the advantage of passing contiguous gene segments from both parents to both child solutions. By preserving these segments (rather than just randomly changing the permutation for example), the operator increases the likelihood of passing good segments (i.e. good partial packing orders) to the child solutions. For mutation, the algorithm randomly selects two objects in a chromosome and swap their positions, thereby making a small change to the packing order.

Results

Fig. 4-17 Shows the results from the comparison between the hyper-heuristic algorithm and the single heuristic order optimisation algorithm for the 4 placement heuristics. For this test, the student ran the hyper-heuristic packing algorithm 40 times using the Jakobs1 dataset (with a container width of 100 pixels). The student then ran the single heuristic algorithms 40 times for each placement heuristic and compared the results to the hyper heuristic. For both test cases, the objects were allowed to rotate in 90-degree increments when being packed. Both the hyper heuristic and order optimisation GA were run with a population size of 50 for 100 generations.

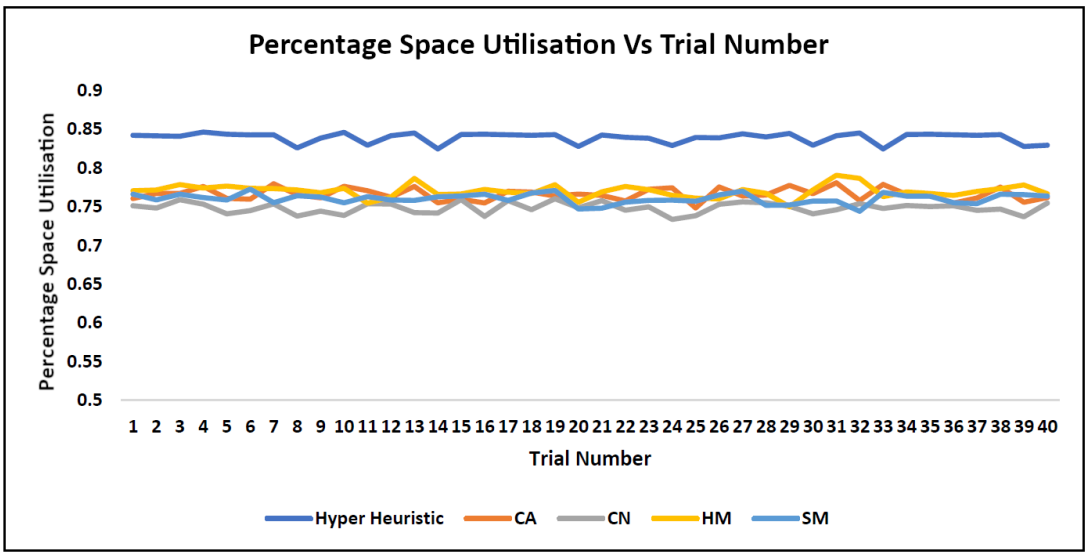


Fig. 4-17. Percentage space utilisation for hyper-heuristic algorithm and single-heuristic algorithms over 40 trials. CA – Contact Area, CN – Contact Number, HM – Height Minimisation, SM – Seat Minimisation. Figure adapted from [173].

From Fig. 4-17 It can clearly be seen that the hyper-heuristic approach has outperformed the single placement heuristics for every trial. The student also noted that, from the single placement heuristic trials, contact area and height minimisation performed the best on average (with contact area performing marginally better than height minimisation), followed by seat minimisation and contact number last. It was also noted that the reason for such poor performance with contact number was likely due to the nature of the shapes in the dataset, with all the shapes having flat sides with minimal curvature. As such, the student was advised to re-run the hyper-heuristic algorithm with some curved shapes added to the dataset. The goal of this was to examine the optimised placement heuristics assigned to the objects to see if there was any pattern between the assigned placement heuristics and the types of shapes being packed. The student added 16 circles to the dataset (9 large and 7 small ones) and then re-ran the hyper heuristic algorithm under the same conditions as before, producing the result shown in Fig. 4-18.

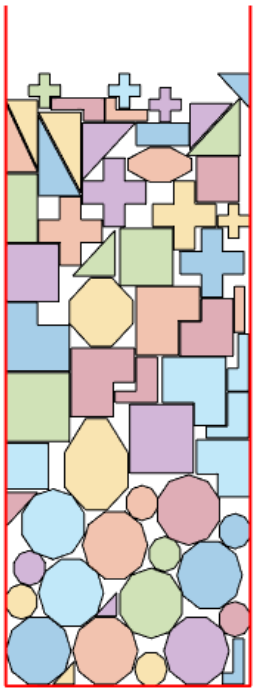


Fig. 4-18. Jakobs1 dataset with circular shapes added. Image from [173].

From this experiment, they noted that, after heuristic optimisation, all the circles were assigned the contact number placement heuristic and were grouped close together in the packing structure (as evident from Fig. 4-18). They also noted that many of the larger, flat sided objects were placed using contact area maximisation. This had the effect of producing interlocking of shapes with complementary edges (e.g. the 'L' shaped objects which nest together well), as well as producing a wall building effect where many of the flat sided objects were stacked against the side of the container (as can be seen in the left side of the container in Fig. 4-18). This suggests that shape properties play a large part in determining which placement heuristics are most appropriate to pack them. Given the limited scope of this study, this topic is considered worth exploring in greater detail in future work.

4.2.3 Multi-Container Packing

Based on the successful application of hyper heuristics in this study, the integration of hyper-heuristics with order optimisation for multi-container packing was subsequently investigated. The remainder of this section describes this extension.

In this study, the goal of optimisation was to find both an ordering for a set of objects, and a placement heuristic for each object such that, when the objects are packed in this order

with each object placed according to its prescribed heuristic, the number of containers, N, needed to pack all the objects is minimised.

Let $O = \{o_1, o_2, \ldots, o_n\}$ denote a set of n 2D irregular objects which are to be packed into identical rectangular containers, with width and length denoted as W and L respectively. Additionally, let $H = \{h_1, h_2, \ldots, h_k\}$ denote a set of placement heuristics available to the packing algorithm, where k is the number of placement heuristics available. The optimisation variables can thus be defined as a 2-tuple set: $\Omega = \{(o_1, h_i^1), (o_2, h_i^2), \ldots, (o_n, h_i^n)\}$ where $i \in 1, \ldots, k$. Each entry in the set Ω contains an object $o \in O$ with an associated placement heuristic $h \in H$ which is used to pack that object.

Assuming the objects are packed in the order that they appear in Ω, the goal of optimisation can be stated as a desire to minimise N by optimising 1.) the order of objects in the set Ω (by changing the permutation of the pair entries in the set), and 2) the choice of placement heuristics used to pack each object (by changing the index i of each h associated with each object in Ω), subject to the following constraints:

1. No overlap between the packed objects.
2. All packed objects are within the container boundary.

Using N directly as the optimisation objective will result in tied solutions for cases where there are multiple solutions with the same number of containers. Hence, the same objective as in [176] is adopted, which aims to maximise the percentage usage of each container. This is expressed as:

$$F = \frac{\sum_{j=1}^{N} U_j^2}{N} \tag{1}$$

Where U_j is the utilisation ratio of each container $j \in 1, \ldots, N$ and is defined as:

$$U_j = \frac{\sum_{m=1}^{n_j} a_m}{LW} \tag{2}$$

Where a_m is the area of object $m \in O$.

The benefit of this metric is that it places a higher score on packing solutions where most of the containers are well packed. For example, consider a simple case with two packing solutions, A and B. Both packing solutions have two containers, however in solution A, both containers have a utilisation score, U, of 0.5 whereas in solution B, one container has a U score of 0.9 and the other has a U score of 0.1. Based on the number of containers, the

solutions are the same. However, if the F scores are calculated for both cases, solution A has a score of 0.25, whilst solution B has a higher score of 0.41.

To pack a set of objects, the algorithm begins by opening a single container into which the objects are placed one by one starting with the first object in the list. Each object is placed according to the placement heuristic assigned to it. If the next object in the list cannot be packed into the current container, the container is closed off and a new container is opened. The algorithm then continues packing the objects into the new container until the next object cannot fit. This process repeats until all the objects have been packed.

The optimisation of the packing order and heuristics for each object is performed by a genetic algorithm. The population of the GA is made up of individual solutions, or 'chromosomes', where each chromosome is equivalent to an instance of Ω and the number of chromosomes is set by the user. Each gene in a chromosome is equivalent to a 2-tuple in the set Ω and contains one object and a heuristic to pack it. The order of the genes determines what order the objects will be packed. In each generation, the GA employs crossover and mutation operators to create a new population of solutions ('offspring') for the next generation. For selecting parent chromosomes for crossover or mutation, tournament selection is used. When generating the new population, the algorithm will generate a random number between [0,1]. If the number is less than the user defined crossover rate, the algorithm performs crossover (producing two offspring), otherwise the algorithm performs mutation (producing a single offspring). This process iterates until a full population of new solutions has been generated.

For crossover, Davis's order crossover is used to generate child solutions with different gene permutations. It is worth noting that when performing crossover, the child solutions will have different gene permutations, however the crossover operator does not alter the placement heuristics assigned to each object in a gene. As such, each gene in a child chromosome will retain the placement heuristic it possessed in the parent chromosome.

For mutation, the operator selects two different genes at random from a parent chromosome and swaps their positions. The operator then randomly changes the value of i for each h in the two selected genes to a different value of i from $1, \ldots, k$, thereby changing the placement heuristics use to pack those objects. The GA also utilises elitism to retain a small portion of the best solutions from each generation to carry over to the next generation. In doing so, the best solution found is retained throughout the GA run.

For packing the shapes into the containers, DigiPac is interfaced with MATLAB. The placement heuristics used by the algorithm were the same four heuristics outlined

previously (height minimisation, seat minimisation, contact area maximisation and contact number maximisation). The initial population of solutions comprises randomly ordered solutions with the heuristics assigned to each object likewise generated at random. For all the tests in this study the algorithm was run for 100 generations with a population size of 50. The crossover rate was set to 0.5, the number of elite solutions was set to 5 and the number of solutions for tournament selection was set to 10.

For testing the algorithm, 5 irregular-object benchmark datasets were selected, all of which are available from [174]. The datasets are shown in Table 4-1 along with the number of objects in each set (n), and the permitted rotation angles for objects as given by the original authors of each dataset.

TABLE 4-1. Datasets with number of objects and given rotation angles.

Dataset	n	Given Rotations
Fu	12	[0, 90, 180, 270]
Albano	24	[0, 180]
Swim	48	[0, 180]
Poly5b	75	[0, 90, 180, 270]
Shirts	99	[0, 180]

The results are benchmarked against the results for the same datasets used in [176] for the 'Nest-MB' (Medium Bins) instances, under the same container size conditions. For the Nest-MB instances, the authors of [176] define the container dimensions as $W = L = 1.5m_d$, where m_d represents the maximum dimension (length or the width) across all the objects in an object set, in their given initial orientation. In this study, rather than scaling the containers in proportion to the shapes (as in [176]), fixed container dimensions of $W = L = 100$ pixels are used, and the shapes in each dataset are scaled accordingly.

For the rotation angles of the objects, two separate trials were run on each set of shapes; one where the rotations were the same as the given rotations and another where the shapes were permitted to fully rotate in 1-degree increments. In [176], the authors also test packing the shapes using the given angles as well as using free rotation. Within the context of their paper, free rotation refers to the fact that objects can have any angle between 0 and 360 degrees, however this does not mean that objects are continuously rotated when being packed. Instead, the authors use a pre-processing step before packing each object to identify promising angles based on edge alignment (aligning edges of the object to be packed with flat edges of previously packed objects and the container boundary). They then

select the top three rotation angles which maximise edge alignment, and test packing the shape for each angle.

Results

Table 4-2 lists the results of this study along with the results obtained in [176] for the same data sets, both of which are averaged over multiple runs. Fig. 4-19 shows an example of a packed structure obtained using full rotation (specifically, the best result obtained for the 'Albano' dataset). For the data in this study, the algorithm was run 10 times for each dataset in both cases (given rotation and full rotation). The best results for each dataset are highlighted in bold.

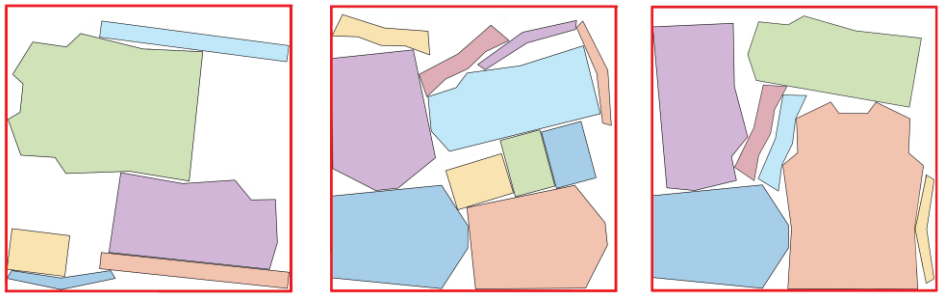


Fig. 4-19. Best packing result for the 'Albano' dataset.

TABLE 4-2. Comparison of results from this study with literature.

	This Study				Results From [176]			
	Given Rotations		Full Rotation (1°)		Given Rotations		Free Rotation	
Dataset	N	F	N	F	N	F	N	F
Fu	4	0.421	4	0.436	4	**0.443**	4	0.440
Albano	3	0.522	3	**0.551**	3	0.480	3	0.510
Swim	5.9	0.334	5.3	**0.404**	5	0.397	5	0.390
Poly5b	8.7	0.371	8	0.439	7	0.478	7	**0.480**
Shirts	9	0.457	8.8	0.520	8	0.518	8	**0.570**

Looking at the results in Table 4-2, the proposed algorithm performed better on all cases using full rotation in 1-degree increments when compared to using the given rotation angles. This is unsurprising as using full rotation allows the algorithm to find more potential sites for each object. When comparing the results from this study (using full rotation) with the results from [176], the algorithm outperformed theirs for the 'Albano' and 'Swim' datasets and achieved close results for the 'Fu' dataset. For the datasets with a higher number of objects ('Poly5b' and 'Shirts'), the algorithm performed noticeably worse. Compared to the algorithm in [176], which uses an allocation algorithm to allocate subsets of objects to

different containers (which are then optimised individually), the algorithm in this study aims to solve the problem in a more global fashion by optimising the entire object set, rather than splitting it into subsets. The result is that, in theory, the proposed approach can achieve better results as it is better able to explore the solution space, however the downside is that the solution space quickly becomes very large for an increasing number of objects.

The number of combinations for the approach in this study can be calculated as:

$$n_{combinations} = n!\,(k^n) \tag{3}$$

Where $n!$ is the number of ways to order a list of objects, n, and k^n is the number of heuristic combinations for k heuristics with n objects. Fig. 4-20 shows a plot of the number of combinations (calculated using Equation 3, with k = 4) plotted against the number of objects, with the 5 datasets used in this study marked with coloured lines.

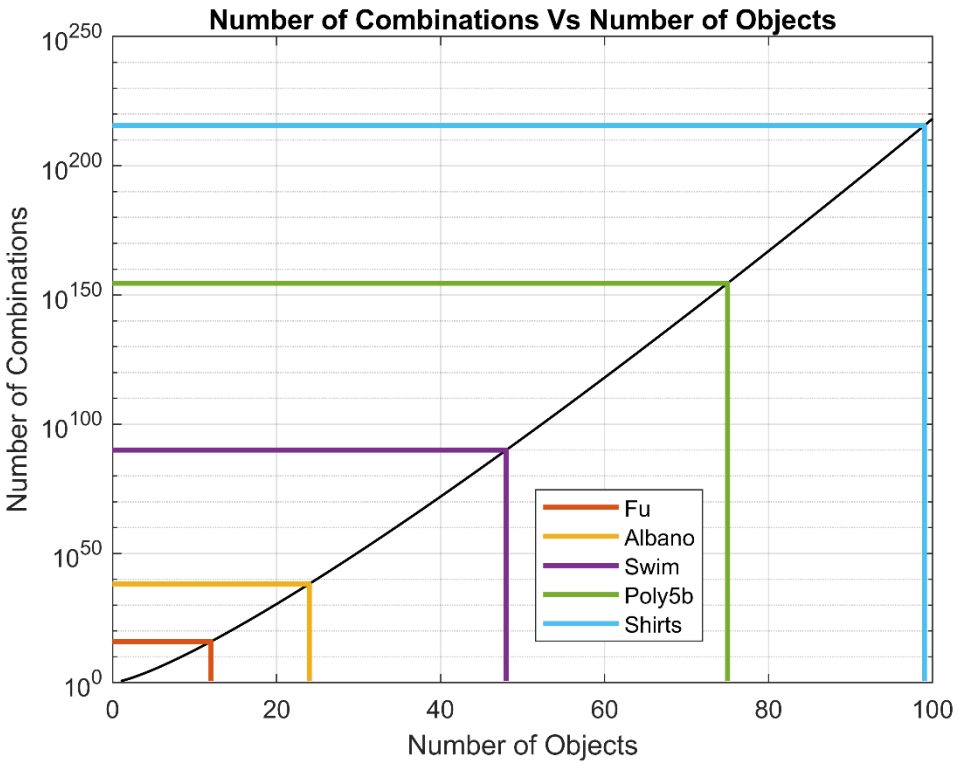


Fig. 4-20. Plot of the number of combinations Vs the number of objects to pack, with the 5 datasets used in this study shown by the coloured lines.

From Fig. 4-20, it can clearly seen that the number of combinations grows very quickly with the number of objects in the dataset (primarily due to the $n!$ term in Equation 3). Consequently, whether the proposed algorithm achieves good solutions or not will depend heavily on both the quality of the initial solutions in the GA, and on the number of

generations the GA is allowed to run for. For example, for the 'Shirts' dataset, the best F score achieved over the 10 runs by the algorithm was 0.593, which is better than the average score (0.570) for the same dataset in [176]. However, this was only achieved in one of the runs, with the rest producing worse results. As such, future work needs to be done to reduce the number of combinations for the problem so the algorithm can consistently achieve good solutions without requiring very long run times or multiple runs.

Regarding computation times, the algorithm took much longer to run than the algorithm in [176]. The authors of [176] provided the run time, averaged over all datasets in 'Nest-MB' ,as 116 seconds. For the full rotation case, the average run time in this study was 217 minutes. This longer run time can be attributed to the fact that the proposed approach tests many more packing orders and rotation angles for objects when compared to [176]. In [176] the authors keep the order of objects fixed from largest area to smallest and only test a small number of different orientations. Using a fixed order with only heuristic optimisation was tested with the algorithm in this study, however it was found to produce poor results. Using a fixed order works well when the object set is pre-allocated and the containers are optimised individually (as in [176]), because for each container, large objects end up at the bottom of the container with the smaller objects packed in the gaps. With the approach used in this study however, it was found that keeping the order fixed from largest to smallest area resulted in the first few containers all having only large objects (with many gaps between them) and the last container having many small objects (which only occupied a small portion of the container).

4.2.4 Concluding Remarks: Hyper-Heuristics for Container Packing

Based on the two short studies into hyper heuristics presented above, several points can be noted:

- The proposed hyper heuristic strategy (with the four placement heuristics) outperforms single-heuristic based packing with the individual heuristics, verifying past observations in literature that no placement heuristic is universally optimal in producing good packing arrangements.
- The properties of shapes are an important factor in terms of what placement heuristic is best for packing them, highlighting the importance of problem specific heuristic selection.
- From the multi-container algorithm, it was found that the proposed global hyper-heuristic approach can outperform literature for some small problem instances, but

that the search space quickly becomes very large with an increase in the number of shapes, making it difficult for the algorithm to consistently produce good quality solutions.

Ultimately, the biggest problem regarding hyper-heuristic packing optimisation is that without an intelligent selection strategy for selecting low level heuristics, many different combinations must be tried to find a good one (e.g. via metaheuristic optimisation). The problem with this is that it increases the computation times, making such algorithms unsuitable for problems where computational complexity needs to me kept to a minimum.

Future Work

Based on these observations, there are several areas that should be focused on for future work:

Object placement selection strategy: One promising direction for future work would be to investigate the relationships between shape properties and the placement heuristics used to pack them with the aim of developing an intelligent selection strategy for deciding what placement heuristic to use to pack a shape. This would involve developing a set of descriptive shape features (such as curvature, aspect ratio, symmetry, convexity, etc.) and then developing testing datasets with different shape types based on these features. The single container hyper heuristic algorithm used in Section 4.2.2 could then be run on the different shape datasets, with the final placement heuristics for each shape after optimisation being recorded. After recording the data, cluster-based analysis could then be used to try and find any links between the shape properties and the placement heuristics used to pack different shapes with these properties.

To take this a step further, it would also be useful to analyse the features of the partially packed pile itself (e.g. the curvature of the surface of the pile, the variation in the pile height and the distribution of void spaces in the pile), to try and develop a more advanced selections strategy which is able to consider both the features of the shape to pack as well as the state of the partially packed pile when making choices for the selection heuristic. Whilst prior research into this has been conducted by the research group who authored the papers [165-170], the problem with their approach is that their choice of problem state descriptors focus primarily on the state of the unpacked shapes, rather than the geometric properties of the partially packed pile itself.

It can be argued that focusing solely on the shape to be placed in isolation, without considering the geometric properties of the partially packed pile, may still lead to

suboptimal placement. As a simple example, a shape with large flat sides could be assigned contact area maximisation as the placement heuristic. However, if the pile itself, for example, has a highly curved or uneven surface, there may not be any locations where good contact area can be achieved. I.e. the choice of placement heuristic for the shape may not compliment the geometry of the existing pile.

Rotation angle selection strategy: Currently, with the existing search process in DigiPac, every time a shape is to be packed, it must be translated across the entire packing space in discrete steps (with overlap checking performed at each step), for every allowed orientation. As an example, if shapes are allowed to rotate in 1-degree increments, each time a shape is to be packed (in 2D), the scanning process must be repeated 360 times (for each allowed orientation). It therefore follows that, the finer the angle resolution, the higher the computation time will be for packing the shapes. A simple solution to this problem (as is often used in most existing packing algorithms), is to limit the rotation increments to large values (e.g. 90-degree increments). However, this runs the risk of the algorithm missing potentially good locations for placing the shapes.

As such, another promising area for future work would be to investigate using rotation selection schemes to intelligently limit the rotation angles for objects to more 'promising' rotations. For example, in the benchmark study ([176]), the authors use a rotation selection scheme which considers the flat edges of the object to pack as well as the edges of the packed objects and container to calculate rotation angles for the object which promote edge alignment between the packed objects or container wall. In [177], the authors use principal component analysis to identify convex features on the object to pack and use this to calculate rotation angles which aim to promote good nesting between the object and the packed pile.

Future work should focus on developing additional rotation selection schemes and then trying to link them with the different placement heuristics in the hyper-heuristic algorithm. For example, a rotation selection scheme which favours matching flat edges (such as in [176]) could pair well with contact area maximisation since aligning long flat edges is likely to increase surface contact. The broader idea is to identify complementary pairings between rotation selection strategies and placement heuristics to promote optimal packing whilst minimising computation time when searching for packing locations.

Integrating hyper-heuristics with allocation-based multi-container packing: As stated, one of the problems with the proposed multi-container hyper heuristic algorithm is the factorial growth rate of the search space with an increase in the number of objects to pack

(resulting from the $n!$ term in Equation 3). In comparison, one of the main benefits of the benchmark algorithm is that it splits the multi container packing problem into sequential single container optimisation, where a separate optimisation process is used to allocate subsets of objects to each container and the packing order is fixed form largest to smallest. This approach largely reduces the computational burden by removing the need to evaluate many packing orders for the object set.

Another promising research direction would be to integrate the proposed hyper-heuristic framework into this allocation-based multi-container algorithm. The benefit of doing so is that in theory, it would remove the need for placement heuristic optimisation (by using an intelligent process to allocate placement heuristics) and the need for order optimisation, whilst allowing the algorithm to leverage a hyper-heuristic framework to improve packing.

Additionally, one of the other benefits of this proposed framework is that it decouples the problem of selecting which part to pack next from the hyper-heuristic decision-making process. This contrasts to previous approaches, such as the work conducted by the authors of [165-170], where the selection policy must select both the next shape to pack and how to pack it. The problem with this is that incorporating the decision on which part to pack next increases the size of the decision space. For example, in the work conducted by the authors of [165-170], they present several different placement heuristics with several different selection heuristics for picking which object to pack next, giving a total of 40 different combinations (i.e. 40 different decisions) which the algorithm can select from. In contrast, by decoupling the object selection process from the hyper-heuristic packing, the proposed future hyper-heuristic algorithm would have a much smaller decision space (of 4 placement heuristics with the current implementation), helping to simplify the decision-making.

Another problem with incorporating the decision of which part to pack next into the hyper-heuristic framework is that care must be taken to ensure the algorithm doesn't become too greedy (i.e. selecting shapes that prioritise short term gains, like selecting shapes that fit easily into the current pile, over long-term gains, like minimising the overall number of containers). One of the benefits of the allocation strategy used in the benchmark algorithm is that it uses a selection strategy which aims to minimise greedy behaviour by favouring larger objects (for a more detailed explanation on this, see section 4.3.1), helping to maintain a more globally informed packing strategy.

Integrating the proposed hyper-heuristic framework into this process would therefore provide a more balanced hybrid strategy, with the allocation step controlling object

selection with global awareness, while the hyper-heuristic focuses solely on making adaptive placement decisions. This separation of processes would in turn help keep the decision space more tractable, help reduce greediness in part selection, and would allow the placement strategy to adapt to the local geometry of each container.

Adapting the work to 3D: As a final point to note, whilst applying the proposed framework to 2D packing problems would still be a novel contribution to 2D packing literature, extending the approach to 3D would offer greater novelty. Given the very limited research on hyper-heuristics in 3D packing, a 3D implementation of the algorithm could yield valuable insights and help significantly advance the state of the art in this area.

4.3 Allocation-Based Multi-Container Packing

Whilst the focus of the previous section was on optimising object order and placement heuristics using a global optimisation strategy, the computational cost of this approach proved to be a significant limitation, particularly for the large object sets. In contrast, the benchmark study achieved strong results with substantially lower computation times by using an allocation-based strategy.

Given that the proposed approach for solving the cutting and packing problem requires the evaluation of many different cutting patterns produced by the GA, this presents a problem for the hyper-heuristic algorithm. The hyper heuristic algorithm tested in the previous section is already computationally expensive, due to the need to optimise the packing order/placement heuristics. As such, incorporating such an approach into the 3D cutting and packing algorithm would likely lead to excessive computation times. Another issue which was noted in the multi-container hyper heuristic study was the inconsistency of the algorithm, particularly for larger object datasets. The problem with this is that, again, due to computational complexity considerations, it is not feasible to run the hyper-heuristic packing algorithm on each set of cut parts multiple times to obtain a good result.

Given these facts, the decision was made to evaluate the efficacy of the allocation-based packing strategy from [176] using the 4 placement heuristics (height minimisation, seat minimisation, contact area maximisation, contact number maximisation), implemented in a single heuristic fashion, with the aim of:

1. Examining how the allocation-based packing approach (implemented with the DigiPac placement strategy) scales with increasing number of objects (both in

terms of quality of solution and consistency), and how it compares to the hyper heuristic algorithm presented in the previous section.

2. Conducting a more detailed comparison of the different placement heuristics (to investigate the efficacy of the different placement heuristics under different levels of problem complexity).

4.3.1 Partial Bin Packing Algorithm

The following section outlines in detail the general process for the allocation based packing algorithm proposed in [176].

Let $O = \{o_1, o_2, \dots, o_n\}$ denote a set of n 2D irregular objects which are to be packed into identical containers (of equal width and height), with volume denoted as V_c. All the objects in O are small enough to fit into the containers and the objective is to minimise the number of containers, N, needed to pack all the objects in O, subject to 1.) no overlap between packed objects, 2.) all packed objects being entirely contained within their respective containers. A solution to the problem can thus be defined as a set of containers $C = \{c_1, \dots, c_N\}$.

Partial Bin Packing (PBP) is an allocation-based packing approach which aims to solve the multi-container packing problem one container at a time. PBP works by assigning a subset of objects from O to an open container c_i, $i \in 1, \dots, N$, whilst keeping the unpacked objects from O in reserve. The subset is selected such that the sum of the allocated object's volumes is maximised without exceeding the available volume in the container, V, where $V = V_c$ for an empty container. The allocated objects are then placed one-by-one into the container, c_i, in decreasing order of volume. If all the objects are successfully packed, the container is closed, a new container is opened, and the process of allocation and packing is repeated for any remaining unpacked objects. If all the allocated objects do not fit into the container (as is often the case), the assignment is mended until the packing becomes feasible. A simplified flowchart of this process is shown in Fig. 4-21, with the mending loop highlighted.

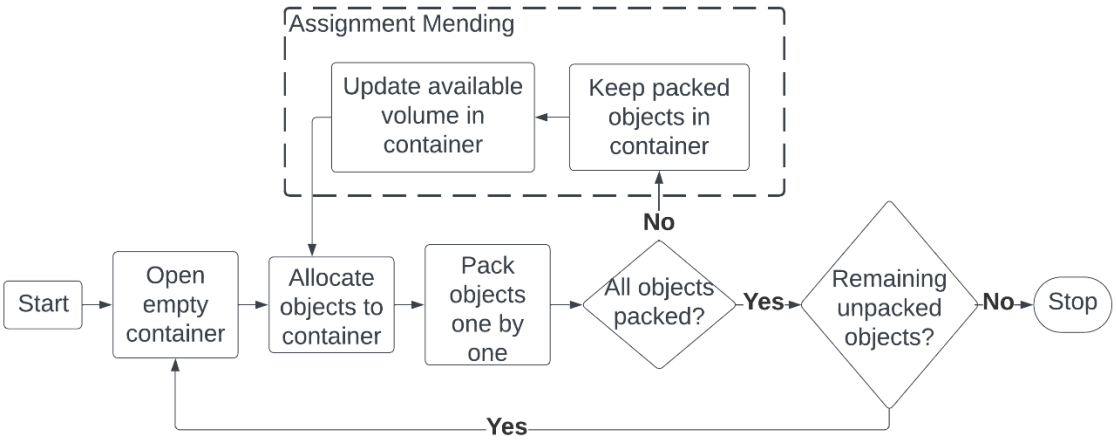


Fig. 4-21. Simplified overview of PBP packing process.

When attempting to pack an allocated subset one-by-one, the mending procedure will trigger when the packing algorithm encounters an object which cannot be packed into the container. When this occurs, any successfully packed objects are kept in the container, the object which failed to fit is placed into a temporary exclusion set, and any remaining unpacked objects in the allocated subset are returned to the main object pool O. The algorithm will then update the available volume in the container (V) based on the volume occupied by the packed objects, and the allocation process will run again to select another subset from O to try adding to the partially packed container. The process of mending the assignment and packing the parts will repeat until either all allocated objects are successfully packed, or there are no remaining objects left to try adding to the container.

Allocation Algorithm

The allocation problem itself is a 1D knapsack formulation where objects are selected to maximise the sum of their volumes without exceeding the available volume in the container. The problem, however, with using the sum of object volumes directly as the optimisation objective is that it results in 'greedy' solutions, where smaller objects will be packed earlier on in the solution process, leaving larger objects till the end [178]. For example, consider a simple case with 3 objects of volume $o_1 = o_2 = o_3 = 5$ and 3 objects of volume $o_4 = o_5 = o_6 = 3$, which are to be packed into identical containers with volume $V_c = 9$. Using a greedy approach which maximises the sum of volumes will lead to a 4-container solution where the first container holds 3 objects, o_4, o_5, o_6, (with total volume of 9), and 3 containers each holding a single object, o_1, o_2, o_3, (each with volume 5). However, by assigning one of each object type to a single container (i.e. one object with volume 5 and one object with volume 3), it is possible to pack all the objects into 3 containers instead [178].

To help mitigate this issue, the authors of [176] adopt an objective function widely used in 1D multi-container packing, which favours assigning larger pieces [179]. The problem can formally be expressed as follows:

$$Max. \sum_{k=1}^{n} \left(\frac{v_k}{v_{max}}\right)^2 q_k \quad s.t. \tag{4}$$

$$\sum_{k=1}^{n} v_k\, q_k \leq V, \;\; 1 \leq k \leq n, \tag{5}$$

$$q_k \in \{0,1\}, \;\; 1 \leq k \leq n, \tag{6}$$

Where $V = V_c$ for an empty container and $V < V_c$ for a partially packed container, $q_k,\ k = 1, \dots, n$ is a binary variable taking the value 1 when object k is assigned to the container and 0 otherwise, and $v_k,\ v_{max}$ are the volumes of object k and the largest object, respectively.

Detailed Algorithm Procedure

A detailed step-by-step outline for the PBP procedure is given as follows:

Step 0: Initialise: $C = \emptyset, i = 1$

- C contains all the packed containers, initially set to empty.
- i is the container index, initially set to 1.

Step 1: Initialise: $O_{ex} = \emptyset, O_{temp} = \emptyset, c_i = \emptyset, V = V_c$

- O_{ex} is the exclusion set, containing objects to temporarily exclude from O when mending an allocation.
- O_{temp} is a temporary set which contains the allocated objects.
- c_i is container i, initially empty.
- V is the available volume in c_i, initially set to V_c for an empty container.

Step 2: Allocate objects:

- **If** $O = \emptyset$ **AND** $O_{ex} \neq \emptyset$:
  - No objects left in O to try adding to partially packed container.
  - Return objects in O_{ex} to O.
  - Close container c_i and store to C.
  - $i = i + 1$, return to step 1.
- **Else:**
  - Solve the 1D knapsack problem to select a subset of objects from O. Remove subset from O and store to O_{temp}.

Step 3: Input O_{temp} to the packing algorithm:

- **If all objects successfully packed:**
  - Close container c_i and store to C.
  - **If:** $O = \emptyset$ **AND** $O_{ex} = \emptyset$, stop and return C.
  - **Else:** return any items in O_{ex} to O, $i = i + 1$, return to step 1.

- **If an object cannot be packed:**
  - Stop packing, go to step 4.

Step 4: If all the objects cannot fit into the container:

- Store packed objects from O_{temp} to c_i.
- Move object which didn't fit from O_{temp} to O_{ex}.
- Return any other unpacked objects in O_{temp} to O.
- Set $V = V_c - vol\_packed\_objects$,
- Return to step 2.

4.3.2 Implementation and Datasets

For implementation, as with the hyper heuristic algorithm, the PBP algorithm is implemented in MATLAB, with the shape packing performed by DigiPac (interfaced with MATLAB). To solve the allocation problem (Equation 4), MATLAB's in-built GA solver was used (with the number of generations set to 500). The tests were run on the same desktop PC which was used in the hyper heuristic study to ensure a fair comparison between run times. For assessing the quality of the packed solutions, the same metric (F-score) was used as before (Equation 1).

The results are benchmarked against the same packing instance (Nest-MB) from [176] using the same container size as in the previous study (length = width = 100 pixels), however 5 additional shape sets were added to the testing datasets, giving 10 shape sets in total (shown in Table 4-3). As before, the datasets were selected based on the fact that they provide a good distribution in terms of the number of objects to pack, enabling an assessment of how the algorithm performs with an increasing number of objects.

TABLE 4-3. Datasets with number of objects and given rotation angles.

Dataset	n	Given Rotations
Dighe2	10	[0]
Fu	12	[0, 90, 180, 270]
Mao	20	[0, 180]
Albano	24	[0, 180]
Poly2b	30	[0, 90, 180, 270]
Shapes1	43	[0, 180]
Swim	48	[0, 180]
Trousers	64	[0, 180]
Poly5b	75	[0, 90, 180, 270]
Shirts	99	[0, 180]

Regarding rotation, the algorithm was run using the given rotations (as before) and using 10-degree rotation increments as well. During testing, PBP algorithm was tested with the height minimisation placement heuristic on all 10 datasets using 1-degree and 10-degree rotation increments. It was found that using the 1-degree increments only provided an average improvement in the F-score of 0.72% with an increase in run time of 130.34%. Given the large difference in run times between the two angle resolutions and minimal gains resulting from use of 1-degree rotation increments, the decision was made to use 10-degree increments for the comparison with the given rotations.

4.3.3 Results

Tables 4-4 and 4-5 (on the subsequent page) show the results obtained for each dataset for each placement heuristic (averaged over 10 runs). The run time, T, in seconds for each dataset (averaged over the 10 runs) is reported as well.

From Tables 4-4 and 4-5 it is noted that, as with the hyper heuristic algorithm, the tests using the small rotation increment (10-degrees) outperform all the tests using the given rotation angles. Again, this is unsurprising since the use of small angle increment allows the packing algorithm to test more placement locations, allowing it to find better ones. One of the main notable differences between the results here and the hyper heuristic algorithm used in Section 4.2.3 above are the run times. As a reminder, the average run time for the 5 datasets presented in the hyper heuristic study was 217 minutes. In contrast, the average run times (for the 10-degree cases) for packing the 10 datasets was 89 seconds for seat minimisation, 88 seconds for height minimisation, 86 seconds for contact area maximisation and 96 seconds for contact number maximisation, giving a total average run time of 90 seconds overall. This is significantly faster than the hyper heuristic algorithm and more closely aligns with the average run times achieved for the nest-MB instances in [176] (116 seconds averaged over the datasets).

Examining the results for the individual placement heuristics, it can be seen that, based on the average performance metrics (N and F), height minimisation performed the best overall, followed very closely by contact area. The better performance of height minimisation can be attributed to the fact that it naturally seeks to minimise the height of objects in the packing pile, leading to packing arrangements with tighter vertical compaction and more free space on top of the packing pile. In contrast, contact area is more shape dependent, performing well with objects that have large flat surfaces or complementary interlocking geometries. Whilst this can lead to compact local packing arrangements (i.e. nesting of

certain objects), it does not directly seek to minimise the height of the packing structure. As a result, it may lead to packing structures that are locally dense, but less optimal in terms of the overall vertical compaction.

TABLE 4-4. Results obtained for seat and height minimisation placement heuristics.

	Seat Minimisation						Height Minimisation					
	10 Degrees			Given Rotations			10 Degrees			Given Rotations		
Dataset	N	F	T	N	F	T	N	F	T	N	F	T
Dighe2	2	0.246	8	2	0.24	5	2	0.26	8	2	0.258	5
Fu	4	0.4	27	5	0.291	17	4	0.419	26	4	0.401	15
Mao	3	0.284	15	3	0.262	9	2	0.506	13	3	0.262	9
Albano	3	0.496	23	3	0.492	16	3	0.497	22	3	0.489	17
Poly2b	4	0.398	44	4	0.373	36	4	0.389	42	4	0.369	40
Shapes1	7	0.299	121	8	0.225	110	7	0.288	116	7	0.283	93
Swim	5	0.391	83	5	0.378	69	5	0.395	83	5	0.377	70
Trousers	3	0.597	61	4	0.372	34	3	0.587	57	3	0.565	38
Poly5b	7.2	0.488	233	8	0.395	217	7	0.503	217	8	0.413	208
Shirts	8	0.566	279	9	0.418	234	8	0.584	293	9	0.433	208
Avg.	**4.62**	**0.417**	**89**	**5.1**	**0.345**	**75**	**4.5**	**0.443**	**88**	**4.8**	**0.385**	**70**

TABLE 4-5. Results obtained for contact area and contact number maximisation placement heuristics.

	Contact Area Maximisation						Contact Number Maximisation					
	10 Degrees			Given Rotations			10 Degrees			Given Rotations		
Dataset	N	F	T	N	F	T	N	F	T	N	F	T
Dighe2	2	0.284	7	2	0.258	5	2	0.264	8	2	0.258	5
Fu	4	0.414	26	5	0.291	16	4	0.414	27	5	0.291	16
Mao	2	0.508	11	3	0.262	9	3	0.285	15	3	0.262	9
Albano	3	0.498	23	3	0.492	17	3	0.501	24	3	0.489	17
Poly2b	4	0.398	42	5	0.285	37	4	0.393	44	4	0.376	36
Shapes1	7	0.283	93	8	0.243	117	8	0.229	138	8	0.227	96
Swim	5	0.389	83	5	0.379	68	5.4	0.358	82	6	0.31	70
Trousers	3	0.565	38	4	0.413	56	4	0.412	73	4	0.346	41
Poly5b	7	0.503	226	8	0.401	208	7.6	0.458	231	8	0.397	219
Shirts	8	0.556	309	9	0.418	229	8	0.561	316	9	0.42	240
Avg.	**4.5**	**0.44**	**86**	**5.2**	**0.344**	**76.2**	**4.9**	**0.388**	**96**	**5.2**	**0.338**	**75**

Compared to height minimisation and contact area maximisation, seat minimisation performed the third best out of the four placement heuristics. The problem with seat minimisation is that, much like the other placement heuristics, it doesn't seek to minimise vertical height, instead choosing to simply place objects at the lowest available site in the packing structure. This can potentially lead to uneven packing structures, with objects sticking out of the top of the pile, making it more difficult to add subsequent objects. Interestingly, even though seat minimisation shares this same drawback with contact area maximisation, it is outperformed by it. This is likely due to the fact that contact area maximisation encourages more interlocking between shapes, which can lead to naturally more compact arrangements. Given that seat minimisation functions equivalently to the well know bottom-left placement heuristic, despite its widespread use in literature, the results in this study results suggest that it may not perform as well as other placement heuristics for irregular object packing.

Lastly, it is also evident that contact number maximisation performed the worst on average across the test cases. It is worth noting however that based on the simple experiment presented in Section 4.2.2 Fig. 4-18, contact number maximisation showed strong performance when packing circular objects. This suggests that contact number may still be a useful placement heuristic, but only in very specific scenarios (e.g. scenarios involving highly curved or spherical objects where maximising number of contact points can lead to compact arrangements).

TABLE 4-6. Best results from PBP and hyper-heuristic algorithms, and best results from [176].

	PBP (10°)			Hyper Heuristic (1°)		From [176]	
Dataset	N	F	Heuristic	N	F	N	F
Dighe2	2	0.284	CA	-	-	**1**	**0.823**
Fu	4	0.419	HM	4	0.436	**4**	**0.443**
Mao	**2**	**0.508**	CA	-	-	3	0.3
Albano	3	0.501	CN	**3**	**0.551**	3	0.51
Poly2b	4	0.398	CA/SM	-	-	**4**	**0.4**
Shapes1	7	0.299	SM	-	-	**6**	**0.39**
Swim	5	0.395	HM	5.3	0.404	**5**	**0.397**
Trousers	**3**	**0.597**	SM	-	-	3	0.58
Poly5b	**7**	**0.503**	CA/HM	8	0.439	7	0.48
Shirts	**8**	**0.584**	HM	8.8	0.520	8	0.57
Avg.	**4.5**	**0.449**	**-**	**-**	**-**	**4.4**	**0.489**

Note: CA – Contact Area, HM – Height Minimisation, CN – Contact Number, SM – Seat Minimisation

Table 4-6 shows the best results obtained by the PBP algorithm in this study (with the placement heuristic which achieved this result), along with the best results obtained by the previous hyper-heuristic algorithm, and the best results obtained by the authors of [176] (with the best results for each dataset highlighted in bold). As a point to note, the average results for the hyper-heuristic results are omitted since the algorithm was not run on all 10 datasets. Additionally, for the PBP results where two placement heuristics are listed, this was due to a tie in the F-scores between the two heuristics.

Comparing the PBP algorithm to the hyper-heuristic algorithm, it can be seen that the PBP approach is outperformed by the hyper-heuristic for the smaller object number datasets ('Fu', 'Albano' and 'Swim') but is beaten by the PBP algorithm for the higher object number datasets ('Poly5b' and 'Shirts'). Given that the scaling of the objects is consistent across all datasets (with the shapes scaled according to the Nest-MB benchmark used in [176]), this indicates that the observed performance differences are not driven by the object-container size ratios, but rather by the increasing number of objects. This validates the previous theory that the problem with the hyper-heuristic approach is the rapid growth of the search space with increasing object count, making it more difficult to find high-quality solutions.

When comparing the results in this study to the result from [176], it is evident that, whilst the PBP algorithm in this study outperforms theirs on four of the datasets, on average the performance of theirs is better. This is most likely due to the use of voxels in this study, which restricts both the position and the orientation of objects to a discrete grid and discrete set of angles, potentially limiting its ability to find more compact arrangements. In contrast, their approach allows for continuous placement and true free rotation of objects (i.e. objects can take any orientation), enabling the algorithm to find placement sites that the algorithm in this study may miss due to its discrete nature. Furthermore, instead of relying on a fixed placement heuristic to select sites, they compute all the geometrically feasible packing sites where an object can be placed using the No-Fit-Polygon (NFP) and select one that minimises the bounding rectangle of the packing structure. This allows them to evaluate the global impact on the packing structure for each local placement, rather than making a decision based solely on individual objects in isolation. As a result, each object is placed in a way which contributes to the overall compactness of the packing structure, rather than just satisfying a local objective.

While the method presented in [176] performs well in 2D, it is not well suited for the intended application of packing nuclear waste in 3D. The primary issue is that their packing method relies on the NFP concept for finding feasible packing sites for objects. As stated in

the literature review, calculation of the NFP in 2D is a well-studied subject with multiple fast and robust procedures proposed for its calculation. In contrast, extending the approach to 3D is far from trivial, with only a very limited number of studies ([31] and [53]) attempting it. As such, there is no efficient, generalised method for generating NFPs in 3D limiting its practical use in real-world applications.

A final point to note regarding the PBP and hyper-heuristic algorithms presented in this study is the consistency of the PBP approach compared to the hyper-heuristic. It was noted with the hyper-heuristic that it had a tendency (particularly with the higher object number datasets) to produce inconsistent results, leading to average container values (N) which were not all integer values (as can be seen in Table 4-6). In contrast, the PBP algorithm produced much more consistent values across the 10 runs, almost always producing solutions with the same number of containers. This is because the hyper-heuristic uses a global optimisation approach that explores many object orderings and placement strategies, which rapidly increases the search space and makes it harder for the algorithm to consistently converge to the same optimum. In contrast, the PBP approach keeps the packing order fixed and only utilises a single placement heuristic, meaning the only source of variation in the solutions is from the allocation of subsets to the containers. However, since the allocation problem (Equation 4) is much simpler and faster to solve (when compared to the packing order and placement search done by the hyper-heuristic), it tends to converge more reliably to the same (or very similar) solutions across multiple runs.

4.3.4 Concluding remarks: Allocation-Based Multi-Container Packing

Based on this short study, the efficacy of the PBP packing approach has been demonstrated against the order-optimisation based hyper-heuristic approach. Whilst the hyper-heuristic algorithm can produce better solutions for the smaller object number datasets, its performance significantly degrades for larger datasets due to the exponential growth of the search space. In contrast, by fixing the packing order and simplifying the problem to sequential single container packing, the PBP algorithm can produce solutions much faster and to a greater consistency than the hyper-heuristic approach, even outperforming the hyper heuristic approach (both in terms of speed and quality of result) for the large object number datasets.

As a result of these findings, the decision was made to use the PBP approach for the multi-container packing algorithm in 3D for three main reasons:

1. Whilst the hyper-heuristic performs better for the small object number datasets, realistically, nuclear decommissioning is likely to involve the cutting and packing of large structures, potentially leading to many cut parts. As such, a packing approach which shows strong and robust performance for large object datasets is favoured.
2. Given the average run time for the hyper-heuristic algorithm was much larger than for the PBP algorithm, this would be particularly problematic if the hyper-heuristic algorithm was integrated into the proposed cutting and packing approach since the approach requires the evaluation of many cutting patterns.
3. By using a more consistent packing approach (PBP) it also allows for the better linking of the cutting and packing problem. With the hyper-heuristic, it struggled to give consistently strong solutions for anything more than a small number of objects. Given the proposed cutting and packing approach ranks the quality of solutions based on both cutting and packing, this inconsistency could easily lead to cases where a good cutting solution is paired with a suboptimal packing solution, resulting in the combined solution being classed as poor, even though one component is high quality.

Based on the comparison of the different placement heuristics it was found that height minimisation and contact area both performed the best, with height minimisation performing slightly better on average. Based on this finding, the decision was made to use height minimisation for the PBP implementation in 3D. In addition to this fact, height minimisation was also chosen for the following additional reasons:

1. In terms of the average number of containers, height minimisation performed strongly not just for the 10-degree rotation case, but also for the given rotation cases as well, with the average number of containers being 4.8, which is only marginally worse than 4.5 for the 10-degree rotation cases. Given that packing in 3D adds two additional rotation axes, trying to test every orientation for an object in 10-degree increments would be impractical as it would involve having to test 36^3 (46,656) orientations. Given this fact, it is likely that the rotation of objects will need to be restricted to large increments (e.g. 90 degrees). As such, a placement heuristic which can still produce good packing solutions with larger angle increments is desirable.
2. Additionally, one of the benefits of height minimisation is that by seeking to place objects with the height of the object as low as possible in the packing structure, it follows that the objects centre of mass within the packing pile will be minimised as well. The benefit of this is that it should naturally result in more stable packing

configurations. For example, in [55] the authors present a packing algorithm to pack objects into containers such that the objects remain mechanically stable in their resting locations. They use a placement heuristic referred to as ‘heightmap minimisation’ which seeks to minimise the increase in the container heightmap. Although their method differs from the height minimisation approach in this study (in that it uses the heightmap of the entire structure), the underlying principles are similar; both approaches aim to minimise the height of the packing structure to improve compactness and promote stability. Notably, their study also compared the heightmap minimisation heuristic against two others; a bottom left fill approach (equivalent to seat minimisation) and another heuristic referred to as ‘maximum touching area’ (equivalent to contact area maximisation). They showed that heightmap minimisation outperformed the other two placement heuristics in producing stable placements, further supporting the effectiveness of height-based placement strategies for producing stable packing arrangements.

4.4 Disassembly Sequencing for Cut Structures

Of the limited examples where cutting optimisation has been performed in nuclear decommissioning, none of the cutting algorithms consider disassembly sequencing for the structure. By disassembly sequencing, this means finding an order of cut part removal such that each cut part can be removed from the structure following a collision free trajectory, whilst ensuring the structure remains stable. Whilst such considerations may be less critical when cutting is performed manually (since humans are able to intuitively assess and adapt to such constraints in real time), disassembly planning becomes essential in the context of autonomous decommissioning. The long-term vision for the research conducted in this thesis is to develop a fully integrated cutting and packing pipeline for robotic decommissioning to allow for the fully autonomous cutting and packing of nuclear structures. Achieving this will require algorithms that not only optimise cutting patterns, but also generate feasible disassembly sequences to enable safe and efficient part removal by autonomous robotic systems.

The following section details work into this subject as part of an MEng project ([180]) carried out under the supervision of the author of this thesis. The project was conceptualised by the author of this thesis, with the coding also carried out by the author. The student performed the testing and data collection for the results presented in this section.

4.4.1 Background

The topic of disassembly sequencing is one which has received a great deal of attention in literature, particularly within the field of electrical and mechanical component recycling. Within this field, the idea behind disassembly sequencing is to find fast and efficient disassembly sequences for products which have reached the end of their life, with the aim of either recycling individual components, or reusing them in future devices. Examples of this include disassembly sequencing for household electrical items (such as desktop PCs [181], [182], coffee makers [183], electric drills [184], printers [185], [186] and LCD TVs [187-189]), as well as mechanical systems (such as car engine blocks [190], [191], gearboxes [192], [193] or water pump systems [194]). More detailed reviews on this topic can be found in [195], [196].

Whilst these methods have proven effective for such products, they are often not suitable for nuclear decommissioning. For example, they often do not consider structural stability (as there is often no need to do so, especially for small scale electronic devices) or greatly simplify part removal (e.g. by simplifying the removal to a set of removal directions without explicitly considering the part trajectories and whether they can move along the trajectory without collision).

In contrast, nuclear decommissioning presents a different set of challenges. For example, structures can often be large and heavy, where improper sequencing for the removal of cut parts could lead to structural instability and potentially collapse (presenting safety risks). Additionally, with advancements in robotics, there is increased interest in the utilisation of such tools for performing cutting actions autonomously (to reduce the need for human-performed cutting in radioactive environments). In such cases, problems relating to the use of robotics, such as the ability to remove cut parts from the structure without them colliding with the remaining structure, must also be considered.

To date, the concept of disassembly sequencing within decommissioning contexts has received very little attention, with the only example in literature being [197]. In this paper, the authors propose a safety-aware, cut-sequencing algorithm for large structures, motivated by decommissioning tasks such as shipbreaking and aircraft dismantling. Their algorithm starts with a pre-partitioned structure (i.e. the locations of the cuts is pre-determined) where the goal is to decide what order to execute the cuts and where the human cutter should position themselves to minimise the risk of harm from falling components. The safety of each cut is quantified using two criteria; the ‘intensity’ of the motion of all parts after a cut is made and the ‘closeness’ of the cut segments paths as they

fall relative to the position of the cutter [197]. The goal when selecting the sequence of cuts is to maximise safety, by minimising the intensity of motion for the cut parts and by positioning the cutter such that they avoid the path of any falling cut parts.

As a point to note, in their work, they allow parts of the structure to be cut and separated from the structure and then cut further after separation (i.e. a part can drop from the structure and then be cut further on the ground). In contrast, the focus in this study is on the removal of one cut part at a time (calculating linear, collision-free trajectories each part can follow to move it away from the structure), making the process more analogous to robotic disassembly where a robotic manipulator would cut the part while another manipulator grasps the part and removes it. Additionally, stability in this study is treated as a hard constraint, meaning parts are not allowed to be removed if it leads to instability. In contrast, whilst the authors of [197] aim to minimise the intensity of the motion of cut parts, they do not enforce stability as a hard constraint, meaning the structure can be allowed to topple over during the cutting process.

Though not related to decommissioning, two other recent works which are worth noting are [198] and [199]. In [198], the authors propose an algorithm which aims to optimise the speed and efficiency of robotic assembly via a process referred to as ‘assembly by disassembly’, which works by finding an optimum disassembly sequence for a structure and then reversing it for assembly. As with the work in this study, the goal of their algorithm is to minimise the sum of the trajectory lengths for the parts to facilitate faster assembly times. Additionally, they also seek to find collision-free trajectories for the parts to follow and enforce stability of the partially disassembled structure. The main difference however is that their stability method is based on a simplified gravity constraint. Each time a part is removed, they check that all the remaining parts in the structure are all supported by ensuring that no part can move downward along the gravity vector (i.e. that the shortest distance a part can move under gravity is zero). This functions similarly to the connectivity constraint employed in this work, which ensures that the removal of a cut part does not result in any disconnected or unsupported parts, however it does not account for broader structural stability such as tipping or collapse. In contrast, this study treats stability as a hard constraint by performing static analysis to check whether the structure remains physically stable after a part is removed.

In [199], the authors also present an assembly-by-disassembly algorithm which aims to find robotically executable assembly plans. They also seek to find collision free trajectories for the parts to follow and have constraints to ensure that the remaining structure is stable under gravity. Unlike the previous study ([198]) however, they explicitly model gravity using

a physics simulator to ensure physical stability throughout the assembly process, which is more advanced than the static stability approach employed in this study. However, their focus is primarily on feasibility rather than optimality since they do not try to minimise the sum of the trajectory lengths for all the parts, but instead aim to find any valid disassembly plan as quickly as possible. In contrast, the approach in this study treats path length as an explicit optimisation objective with the aim being to find more efficient sequences that minimise the disassembly time.

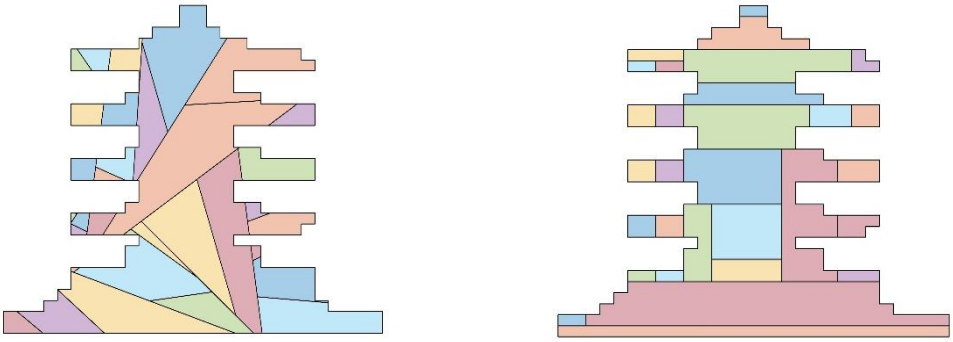
In this study a novel methodology is presented for disassembling randomly cut structures where the goal is to minimise the sum of the trajectory lengths for the cut parts such that the structure remains physically stable. In particular, 3 algorithms are proposed and compared for solving the problem; a greedy search algorithm, a heuristic driven algorithm and a tree search-based algorithm, which are benchmarked against a random search. A novel and fast approach for estimating structural stability is also presented and the performance of the 3 proposed optimisation algorithms is compared under different cutting complexities.

4.4.2 Optimisation Objectives and Constraints

Given a randomly generated cutting pattern applied to a simple 2D test structure (shown in Fig. 4-22), the goal of disassembly sequencing is to find an order of part removal for the cut structure such that:

1. The removal of a part does not result in any disconnected regions (cut parts not connected to the structure), since in the real world, such unsupported parts would fall due to gravity.
2. The removal of a part does not render the structure unstable.
3. There exists a linear collision-free trajectory each part can follow such that it can be removed from the structure without colliding with any of the remaining structure.

The cutting method used to cut the structure is Binary Space Partitioning, which is the same cutting process as outlined in Section 4.1.1. The student tested two different cutting scenarios, cutting with free rotating planes (where the plane rotation angles can take any integer value between [1-180]), as shown in Fig. 4-22 a, and cutting with orthogonal planar cuts (where the rotation angles are limited to [0,90] degrees), as shown in Fig. 4-22 b.



(a) (b)

Fig. 4-22. 2D test structure used in this study cut into pieces by randomly generated cutting patterns. (a) – cutting pattern with free rotating planar cuts. (b) – cutting pattern with orthogonal cuts.

The cutting patterns are initialised with a user-specified number of cuts, with each cut's location and orientation being randomly generated. After generating the random cutting pattern, the cutting pattern is applied to the structure to decompose it into a set of parts. The goal is to then find a disassembly sequence for the parts which minimises the 'cost' of the disassembly, whilst adhering to the 3 constraints. Cost is defined as the sum of the lengths of each cut part's removal trajectory, where each trajectory is a straight line originating from the centroid of the cut part and extending outward until it intersects with the structure's bounding box (Fig. 4-23). This was chosen as the optimisation objective since, within the context of robotic disassembly, minimising all the trajectory lengths will lead to faster disassembly times.

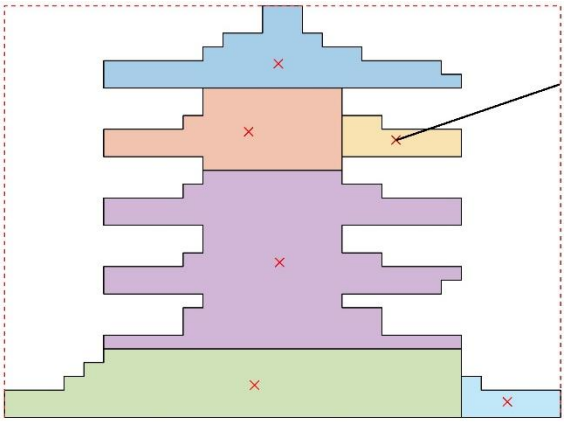


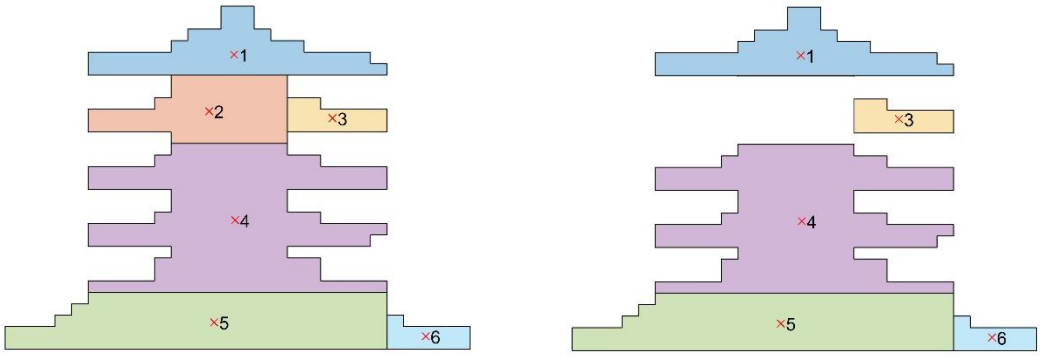
Fig. 4-23. Illustration of 'cost' for a single cut part. Length of the linear trajectory for the cut part (black line) is the cost for that part.

Each time a cut part is to be removed, the algorithm will check to see if it can feasibly be removed by evaluating the three aforementioned constraints. If none of the 3 constraints

are violated, the part can be successfully removed. The three constraints used to assess whether a part can be removed are implemented as follows:

Connectivity

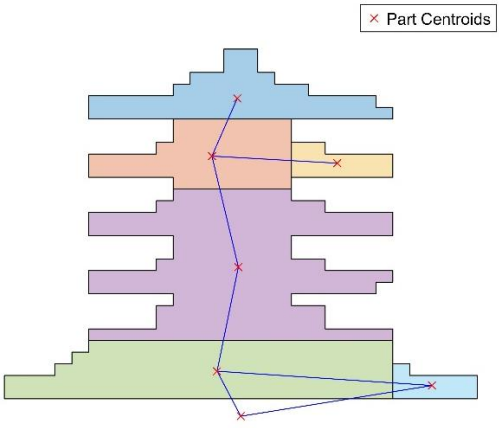
The goal of the connectivity constraint is to check if the removal of a part will result in any disconnected regions in the structure (Fig. 4-24). The purpose of this is to prevent any disconnected, or 'floating', parts being produced due to the removal of a cut part, since in reality, such parts would fall under gravity.



(a) (b)

Fig. 4-24. Example of part removal leading to disconnected regions in the structure. (a) – Cut structure with labelled parts. (b) – part 2 removed, leading to two disconnected regions (part 1 and 3).

The connectivity between parts is represented as an undirected graph, where the nodes correspond to the centroids (centre of mass) of the objects and the connecting edges indicate whether two parts are in contact with one another or not (Fig. 4-25 a). As a point to note, a virtual 'base' is also added to the structure which is included in the connectivity graph (the lowest node in Fig. 4-25 a). This is to ensure that the structure will always be supported at the base (i.e. always connected to the ground).



Part index	1	2	3	4	5	6	7
1	1	1	0	0	0	0	0
2	1	1	1	1	0	0	0
3	0	1	1	0	0	0	0
4	0	1	0	1	1	0	0
5	0	0	0	1	1	1	1
6	0	0	0	0	1	1	1
7	0	0	0	0	1	1	1

(a) (b)

Fig. 4-25. (a) - connectivity graph for a cut structure. (b) - corresponding adjacency matrix.

The connectivity graph for the parts is represented using an 'adjacency matrix' (Fig. 4-25 b). The adjacency matrix is a square binary matrix where each row/column corresponds to a single part. In this matrix, each element indicates whether there is a connection between two cut parts. The matrix is also symmetric, meaning that if part i is connected to part j, both (i, j) and (j, i) entries in the matrix will be marked with a 1. If no connection exists between two parts, the corresponding entry is set to 0. As a point to note, the virtual 'base' for the structure corresponds to the last column/row in the adjacency matrix (column/row 7 in the example shown in Fig. 4-25 b).

Each time the connectivity is checked for the removal of a part, the adjacency matrix for the remaining structure (all remaining parts + the part which is being checked) is found, and the corresponding column/row entry for the part to be removed is then set to 0 (simulating its removal). After removing the part from the adjacency matrix, the algorithm will then check whether all the remaining parts are reachable from one another (i.e. whether each part can reach every other part through a series of connections). If the algorithm finds that some of the parts are no longer reachable from other parts, it identifies this as a disconnection. This is done by keeping track of all the parts which can be visited from any remaining part. If all the other parts cannot be reached from the starting part, the algorithm concludes that the structure is disconnected.

Stability

The goal of the stability constraint is to check whether the removal of a cut part will render the remaining structure unstable. To calculate stability, a simplified representation inspired by the concept of the 'support polygon' in robotics is used. For a legged robot (such as the one depicted in Fig. 4-26), it is considered statically stable on a horizontal plane if, at any given moment, the vertical projection of its centre of mass lies within its support polygon (the convex hull formed around the contact points with the ground) [200].

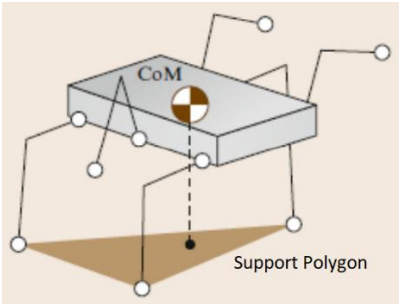


Fig. 4-26. Simple illustration of the support polygon concept for a 6-legged robot. Image adapted from [201].

Regarding the stability of the cut structure, the support polygon principle is applied as follows:

Each time a cut part is removed, the remaining structure must be checked for stability. The algorithm begins by vertically slicing the structure using evenly spaced horizontal slice lines (the red lines shown in Fig. 4-27 a). Then, *for each slice line*, the following steps are performed:

1. **Calculate the support polygon:** Calculate the widest points at which the horizontal slice line intersects with the boundary of the structure (red circles in Fig. 4-27 b). The line between these two points is treated as the support polygon.
2. **Calculate the centre of mass:** Calculate the centre of mass (centroid) for the portion of the structure which exists above the support polygon line.
3. **Project the centre of mass:** Project the centre of mass vertically downward onto the support polygon to obtain the projected centre of mass (Fig. 4-27 b).

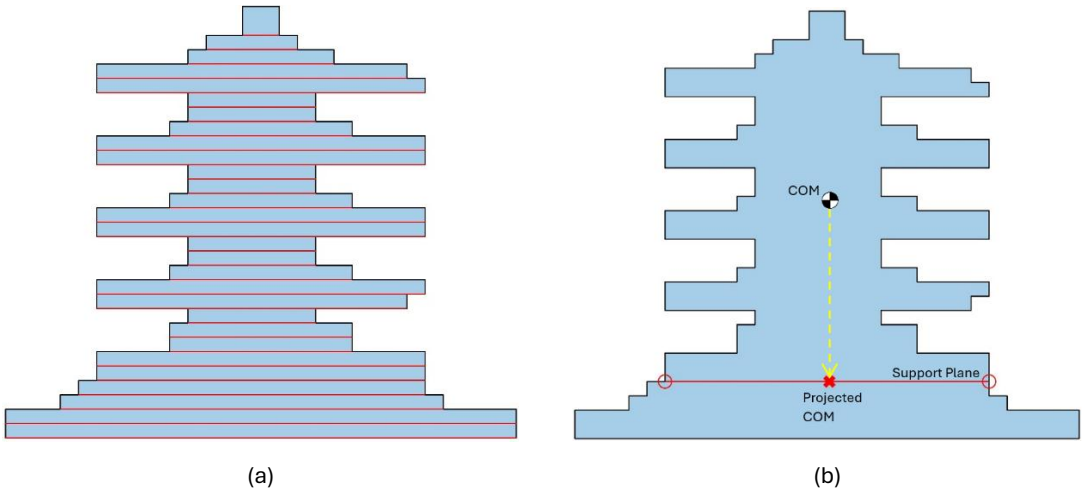
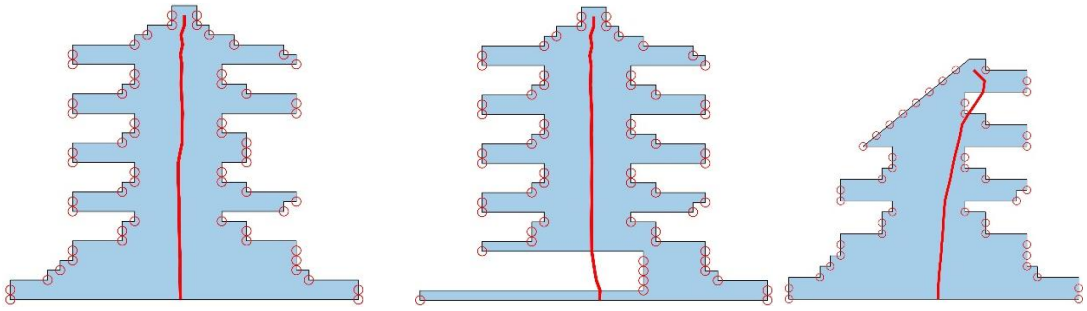


Fig. 4-27. Illustration of how stability is calculated. (a) – structure is sliced using vertical lines. (b) – example of how the support polygon and projected centre of mass are obtained for a single slice line.

Once this operation has been performed for each slice, the projected centre of mass for each slice is checked against its corresponding support polygon to see if it is outside or not. If any of the projected centre of mass points fall outside the support polygon at any part of the structure, the structure is classed as unstable. Fig. 4-28 gives a visual illustration of this concept. In this figure, the projected centre of mass points for each slice have been joined together to form the continuous red line running through the structure. Looking at the left example (Fig. 4-28 a), the red 'stability line' remains between the support polygon points (red circles) at all times, meaning the structure is considered stable. In contrast, with the

two examples on the right (Fig. 4-28 b), due to the removal of parts from the structures, the remaining structures now have large unsupported overhangs. These overhangs cause the support line to leave the support polygon points at several parts of the structure, meaning both are considered unstable.



(a) (b)

Fig. 4-28. Illustrated examples of stable and unstable structures. (a) – stable structure. (b) – unstable structures.

Collision-Free Trajectory

The goal of the trajectory constraint is to check, for a desired part to be removed, whether there exists a linear collision-free trajectory which the part can follow such that it can be moved away from the structure without collision.

The trajectory itself is represented as an angle, θ, around the part's centroid (Fig. 4-29), with the goal of this constraint check being to find a feasible trajectory angle (if one exists).

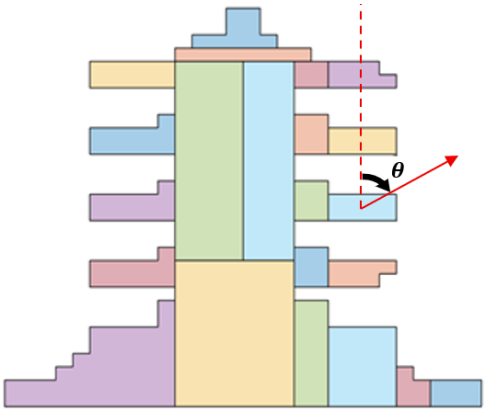


Fig. 4-29. Trajectory for a cut part (blue rectangle) is given as an angle θ with respect to the origin (vertical dotted line). Travel direction for the cut part indicated by the red arrow.

To find a feasible trajectory, the 360° circle (centred on the parts centroid) is sampled in uniformly spaced angle increments (where the angle step size is set by the user). For each

allowed angle, two lines are projected out from the cut part toward the edge of the structures bounding box. The two lines are obtained by calculating the width of the cut part orthogonal to the direction of travel and then drawing two parallel lines, extending out toward the bounding box, originating at the widest points of the part. This creates a virtual 'tunnel' through which the part can move in a straight line such that its widest points will always be in contact with the tunnel walls. Fig. 4-30 illustrates this concept for three different angles, with the red lines showing the direction of travel, the blue lines showing the width of the object orthogonal to the direction of travel, and the black lines showing the calculated travel tunnel.

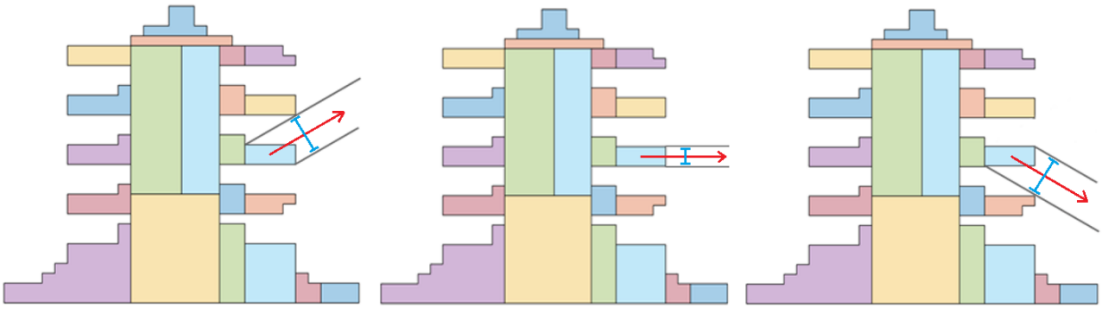


Fig. 4-30. Illustration of trajectory calculation for a cut part (blue rectangle). Red arrow indicates travel direction, blue line indicates width of part orthogonal to the travel direction, black lines show the 'travel tunnel' for the cut part. Image adapted from [180].

After the travel tunnel has been calculated for a given angle θ, an intersection check is run on the two tunnel lines to see if they intersect with any part of the structure. If intersection occurs (i.e. if any part of the structure falls within the travel tunnel lines), it follows that the part cannot be moved along this trajectory without causing collision with the structure. If there is no intersection, the algorithm calculates the trajectory length for that angle and then stores the angle with its associated trajectory length. In addition to checking collision with the structure, the virtual 'base' for the structure (equal to the width of the structure) is also checked for overlap. This is to prevent objects from travelling vertically downward through the floor. Once all the allowed rotations have been tested, the algorithm will select (from the stored solutions) the trajectory with the smallest trajectory length and return this solution from the trajectory function. If no feasible trajectory is found, the algorithm will return -1 from the function, indicating that it has not been able to find any feasible trajectories for that part.

Regarding the choice of methodology for this constraint check procedure, the 'tunnel' approach was used, instead of simply translating the shape along the trajectory whilst checking for overlaps, to reduce computational burden. By using this tunnel approach, it

means that overlap checking only needs to be performed once (on the tunnel lines) for each value of theta. In contrast, using a shape translation approach would require translating the shape along the desired trajectory in discrete steps, running an overlap check each time the shape is moved.

A final point worth noting is that this tunnel approach can theoretically fail if a cut part ends up between the tunnel lines. Referring to Fig. 4-31 as an example, if a cut part (shown by the red square) ends up between the tunnel lines, the intersection check will miss it. However, this situation could only occur if the part between the tunnel lines is disconnected from the rest of the structure, which is not allowed due to the connectivity constraint. As such, it is essential to run the connectivity check before searching for a feasible trajectory, to prevent this from occurring.

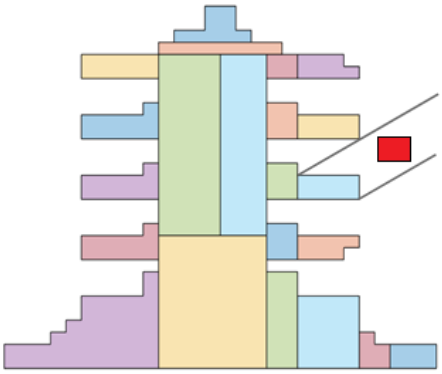


Fig. 4-31. Example of how the overlap checking process can theoretically fail. Red square is between the tunnel lines and would not be detected in the overlap check procedure. Image adapted from [180].

4.4.3 Optimisation Algorithms

To optimise the disassembly sequences, 4 different approaches are implemented and compared. The 4 approaches are as follows:

1. **First Feasible Random Search (FFRS)** – The algorithm will generate random disassembly sequences, stopping at the first feasible disassembly sequence it finds. This is the simplest of the 4 algorithms and will act as a baseline for assessing the performance of the other 3 approaches.
2. **Height Decreasing Search (HDS)** – This is a greedy search algorithm which works by always selecting the highest cut part in the structure which can feasibly be removed at each step. The height of a cut part is based on the y-coordinate of its highest point, rather than the height of its centroid. This process essentially ensures that the structure is disassembled in a top-down fashion. The idea behind this is that it aims to mimic real-world decommissioning practices, such as the dismantling of a reactor pressure

vessel where the structure is typically supported at the base and sides, with components being cut and removed sequentially from the top to help maintain stability and ease of access.

3. **Greedy Search (GS)** – This is another greedy search algorithm which works by always selecting the (feasible) part with the shortest trajectory length at each step. Given that the goal is to minimise the sum of the trajectory lengths for all the cut parts, this algorithm aims to directly optimise this objective, but in a greedy fashion. By locally minimising the trajectory length at each step, the algorithm seeks to build a cost-optimised disassembly sequence quickly (by prioritising short term gains in the total cost), but with no guarantees of global optimality.
4. **Stochastic Tree Search (STS)** – The final algorithm is a stochastic optimisation algorithm inspired by the algorithm presented in [202]. The STS algorithm aims to incrementally build solutions from scratch, with the ability to backtrack in the search procedure if it encounters an infeasible dead end (a point where no more parts can be removed without violating constraints), or it has found a complete solution (in which case it backtracks to try and find a better one).

   The algorithm starts with an ordered list of parts which are sorted from highest part to lowest (based on each part's highest point). The search process is structured as a branching decision tree where each node represents a partial or complete solution, and contains the following information:

   - **X:** a list of parts already removed from the structure.
   - **J:** a list of remaining parts in the structure.
   - **Cost:** the accumulated cost of the current partial solution (i.e. the sum of the trajectory lengths for all parts in X).

   The branches in the tree represent a binary yes/no decision on whether to remove the first part in J. At each node, the algorithm considers the first part in J and evaluates whether it can be removed:

   - **If the part can feasibly be removed**: the algorithm follows the 'yes' branch and removes the part (moving it from J to X and updating the accumulated cost). Simultaneously, it stores the alternative 'no' branch (in which the part is not removed and instead pushed to the back of J) into a temporary storage set for potential backtracking.
   - **If the part cannot feasibly be removed**: the algorithm has no choice but to take the 'no' branch, pushing the part to the back of J to be revisited later. In

this case, the ‘no’ branch is not stored, since there is no alternative path which can be taken.

The tree is explored in a depth-first manner, meaning the algorithm will always attempt to build a complete solution (by selecting the ‘yes’ branch whenever possible) before backtracking. Stochastic backtracking occurs either when the algorithm encounters an infeasible dead end (where no more parts in J can be removed without violating constraints), or if the algorithm has found a complete feasible solution (in which case the goal of backtracking is to explore alternate paths to see if a better solution exists). When the backtracking procedure is triggered, a random node from the temporary storage set is selected and the algorithm resumes its search by taking the path that was not previously selected.

To try and reduce the size of the search space and encourage the algorithm to explore more promising paths, a ‘pruning’ procedure is also implemented. Each time a cut part is selected for removal, the algorithm will compare the current accumulated cost of the partial solution with the total cost of the best-found solution so far. If the current cost exceeds the best cost, the algorithm does not continue down its current path, since doing so would only increase the cost more. In such a case, the stochastic backtracking procedure is triggered to push the algorithm down a different path (effectively ‘pruning’ that branch of the tree).

As a point to note, for both the FFRS and STS algorithm, time limits must be set. With the random search, since it is purely random, it is possible (particularly for complex problems with many cut parts) that it may take a very long time to find a feasible solution. For the STS algorithm, the search process (depth first search with stochastic backtracking) will continue until all possible combinations have been explored. Since this would be computationally impractical (particularly for large problems), a time limit must also be set for this algorithm. For the STS algorithm, if the time limit is reached before all the combinations have been explored, the algorithm will return the best-found solution so far.

4.4.4 Results

As stated in Section 4.4.2, the student decided to test two different cutting scenarios; cutting with free rotating planes (where the rotation angle for the planar cuts can take any integer value between [1-180]) and orthogonal cutting (where the rotation angles are restricted to [0, 90] degrees). For each scenario (free rotation and orthogonal cutting), they

also decided to test the algorithms for increasing problem complexities. They defined problem complexity as 'low', 'medium' and 'high', with low complexity solutions containing 3-5 cuts, medium complexity solutions containing 10-15 cuts, and complex solutions containing 40-45 cuts.

For both scenarios, the student generated 24 different random cutting patterns in total. For the free rotation cutting scenario, they generated 8 low complexity cutting patterns, 7 medium complexity cutting patterns and 9 high complexity cutting patterns. For the orthogonal cutting scenario, they generated 9 low complexity cutting patterns, 7 medium complexity cutting patterns and 8 high complexity cutting patterns. For each cutting pattern generated, the 4 algorithms were run 3 times, with the results averaged over the 3 runs. The cut-off time for the FFRS and STS algorithms was set to 3 hours, and the angle increment for the trajectory sampling was set to 10°.

Fig. 4-32 and 4-33 show the results obtained by the 4 algorithms for the increasing complexity for both scenarios. The plots were created using the raw data found in Appendix A of this thesis (note: data originally presented in Appendices C and D in [180]). For each cutting pattern tested, the best result obtained by the 4 algorithms is considered the 'optimal' result, results worse than this are considered 'suboptimal' and any time an algorithm failed to return a result, the result is listed as 'fail'. The results in both figures show the percentage of solutions which achieved 'optimal', 'suboptimal' and 'fail' for the number of cutting patterns tested in that complexity case.

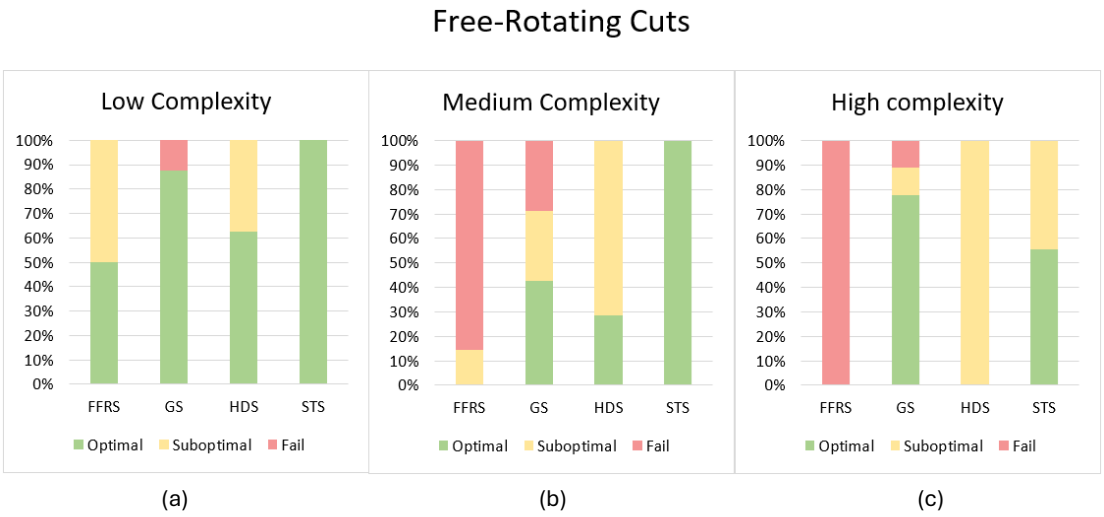


Fig. 4-32. Results for the free rotating cuts scenario. Result show percentage of solutions which were optimal, suboptimal and fail for: (a) the 8 low complexity patterns, (b) the 7 medium complexity cutting patterns, and (c) the 9 high complexity cutting patterns.

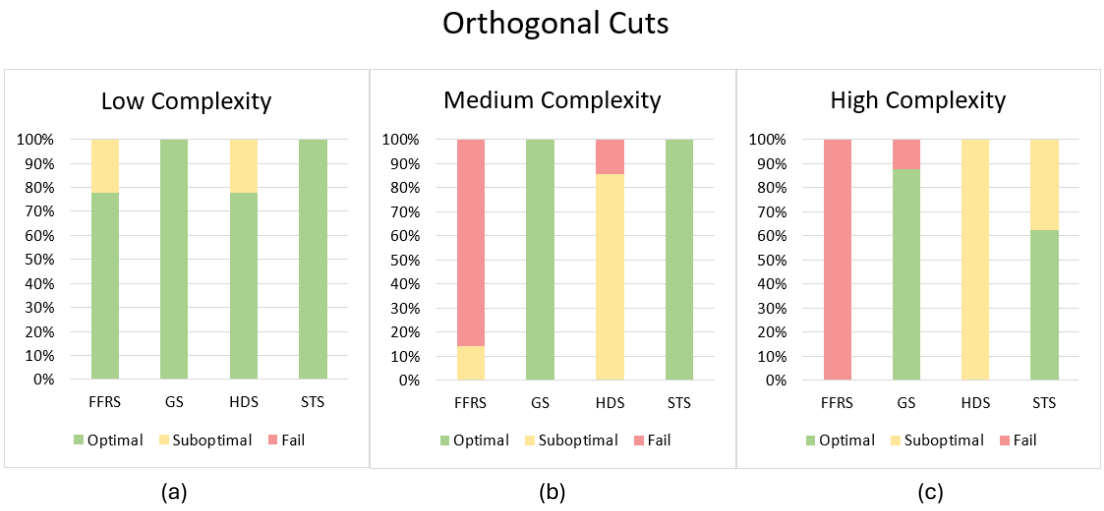


Fig. 4-33. Results for the orthogonal cuts scenario. Result show percentage of solutions which were optimal, suboptimal and fail for: (a) the 9 low complexity patterns, (b) the 7 medium complexity cutting patterns, and (c) the 8 high complexity cutting patterns.

Regarding problem complexity, from Fig. 4-32 and 4-33 it is evident that patterns begin emerging as the complexity increases. For the FFRS algorithm, in both scenarios as the problem complexity increases, the performance degrades very quickly, with the medium complexity cases struggling to find solutions and the high complexity cases failing to find any solution at all over all the runs. Given that this algorithm is purely random and that there are 3 strict constraints which each part must adhere to for it to be successfully removed, this poor performance for increasing complexity is unsurprising.

Regarding the GS algorithm, it is evident that across all the complexity cases, it performed better for the orthogonal cuts when compared to the free-rotating cuts cases. The reason for this is likely due to the fact that with free-rotating cuts, it tends to produce cut parts with a higher variation in both size and geometric complexity, making it more difficult to find removal sequences where all three constraints are satisfied at all times. In particular, the increased geometric irregularity of the parts in the free-rotating case increases the likelihood that a part selected purely in a greedy fashion (based on its trajectory alone) may in fact have a critical structural role in maintaining the stability of the structure. Since the GS algorithm optimises only for cost (while merely enforcing structural constraints like stability), it is likely that it will be more prone to selecting parts which, when removed, lead to instability.

For example, consider the structure depicted in Fig. 4-34. The structure shown is the low complexity (free rotation) cutting case for which the GS algorithm failed to find a solution. For this cutting pattern, if the cut parts shown in Fig. 4-34 (a) are disassembled in a greedy fashion, the disassembly sequence for the parts is [10,7,6,8,5,2,9,4,1,3].

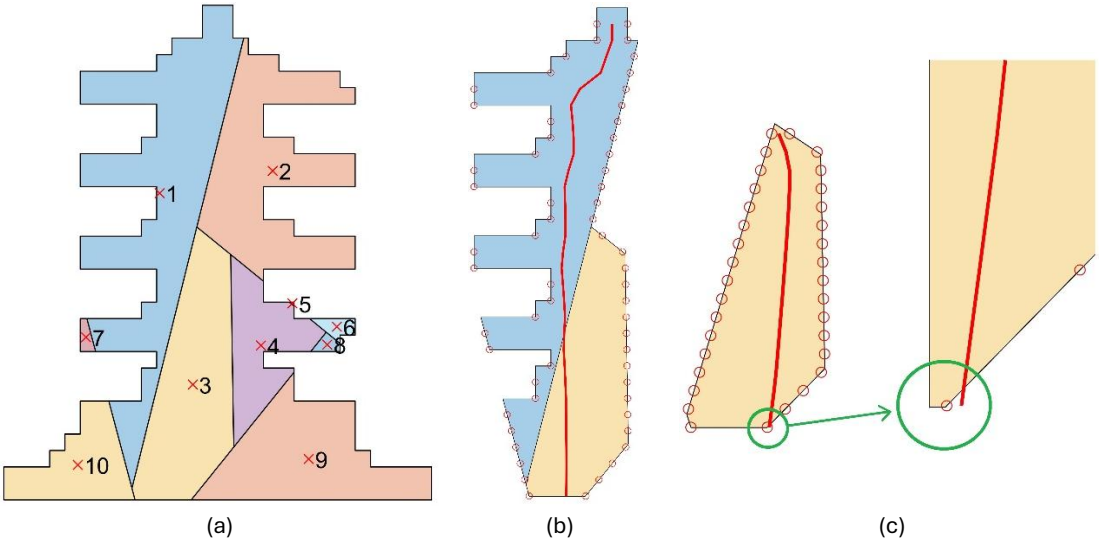


Fig. 4-34. Example of how greedy search can fail. (a) – structure with cut parts labelled. (b) – with part 1 and part 3 remaining, structure is still stable. (c) – once part 1 is removed, structure becomes unstable.

The problem with this disassembly sequence arises when there are only two parts left, part 1 and part 3 (as shown in Fig. 4-34 b). With both parts remaining, the structure is still stable (as shown by the red stability line). However, when part 1 is removed (the large blue part), the remaining part (part 3) becomes unstable (as can be seen in Fig. 4-34 c, where the stability line is marginally outside the support polygon at the base of the part).

This example helps highlight both the issue with free rotating cuts and the issue with using a greedy approach. The production of large, awkwardly shaped parts makes it harder to find a sequence which does not produce constraint violation (in particular, the stability constraint). The use of a greedy search process which favours selecting the cheapest trajectory part each time does not explicitly account for the constraints when making its decision, which can easily result in parts being removed early on that may lead to an infeasible dead end later in the search.

Other than the occasional issue of becoming stuck in infeasible dead ends, it was also noted that the GS algorithm performed well with increases in the problem complexity, even outperforming the STS algorithm in terms of finding optimal solutions for the high complexity cases. The main reason for this is likely due to the fact that it directly optimises the cost objective (by always selecting the part with the shortest trajectory) which allows it to quickly build low-cost solutions without needing extensive exploration of the solution space. Whilst the STS algorithm also seeks to directly optimise cost, it relies on a broader search strategy which explores many more combinations of part removal orders. Since the number of possible disassembly sequences grows factorially with an increase in the number of cut parts, it follows that the search space will become very large very quickly for

increases in problem complexity. In such cases, the exploratory nature of the STS algorithm may become a disadvantage as it can end up spending large amounts of time evaluating suboptimal branches of the tree before reaching a promising region of the search space. In contrast, the GS algorithm's heuristic-driven approach allows it to more efficiently find high quality solutions, even though it does not guarantee global optimality and is prone to becoming entrapped in infeasible dead ends.

Regarding HDS, it showed strong performance across all complexities (for both orthogonal and free rotating cuts) in terms of finding feasible solutions, with only 1 case existing where it failed to find a solution. This strong performance is likely due to the fact that it prioritises top-down disassembly, which naturally leads to more stable configurations that reduce the likelihood of stability constraint violation. However, since the algorithm is also a greedy algorithm (always selecting the highest feasible part in the structure at each step), it is also prone to potentially becoming stuck in an infeasible dead end. This explains the single failure case with the orthogonal cutting, where the top-down strategy may have prematurely removed a key structural component, preventing the remainder of the structure from being feasibly disassembled (due to a violation of the stability constraint). Additionally, since HDS does not directly optimise the cost objective (i.e. it does not consider cost when selecting parts), it is more prone to finding suboptimal solutions (particularly for increasing problem complexity).

Regarding STS, it performed the best overall in terms of finding feasible solutions, with no failure cases being produced in either cutting scenario across all complexities. Additionally, it showed strong performance for the low to medium complexity cases, managing to find optimum solutions every time (for both cutting scenarios). As previously mentioned, the only observable degradation in performance is for the high complexity cases, where it wasn't able to consistently find the optimum result.

The main downside with the STS algorithm compared to the other algorithms is the run time, which scales factorially with the number of parts to remove. Fig. 4-35 shows the run times for each algorithm plotted against the number of disassembly sequence combinations (calculated as $n!$ where n is the number of cut parts) for the low-complexity free-rotating cuts case. Note that the y-axis is logarithmically scaled. Additionally, only runs that resulted in feasible solutions are included in the plot; for example, the data point corresponding to the one low complexity run case where the GS algorithm failed to find a solution has been omitted.

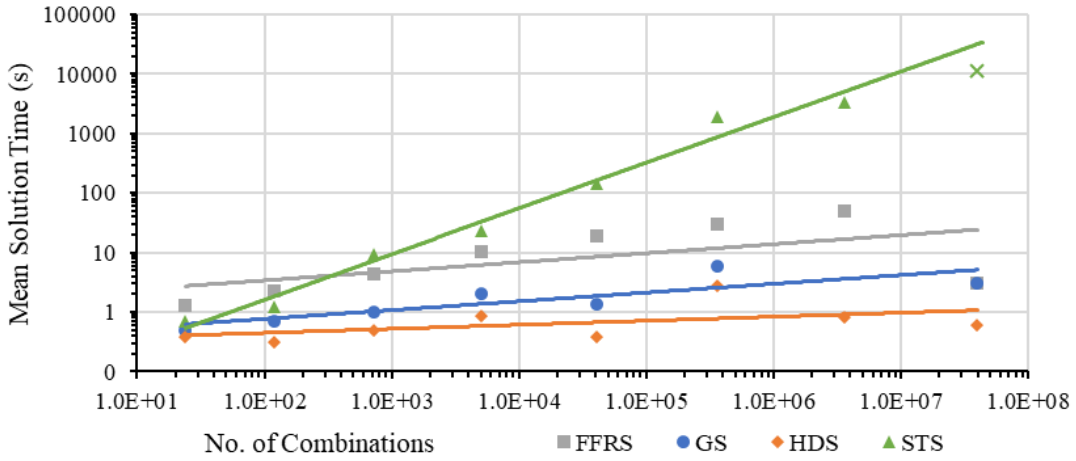


Fig. 4-35. Run times for the 4 algorithms plotted against the number of disassembly sequence combinations for the low complexity, free-rotating cuts scenario. Image adapted from [180].

From this figure it is clear that the run time for the STS algorithm scales exponentially with an increase in the number of combinations to the problem, whilst the run time increases for the other algorithms show linear trends. As a point to note, for the STS and FFRS algorithms, only the run times for the low complexity cases are shown due to the fact that for the medium and high complexity cases, both algorithms always reached the 3-hour cut off time. Additionally, the final data point for the STS algorithm in Fig. 4-35, indicated by the green 'x', is marked as such because the algorithm reached the 3-hour cut-off time before being able to evaluate all combinations. These observations help highlight that, while the STS algorithm is effective at finding feasible (and often optimal) solutions, its computational cost scales poorly with increasing problem complexity, limiting its practicality for larger problem instances without a substantial increase in computational resources.

Regarding the FFRS algorithm, whilst the trend is linear, the run times were generally always greater than that of the GS and HDS algorithms. Given that the FFRS algorithm is purely random by nature (i.e. it does not include any form of heuristic guidance mechanism), the larger run times are unsurprising. In contrast to the FFRS and STS algorithm, the GS and HDS algorithms show strong performance timewise for an increase in the problem complexity. This becomes more evident when examining the run times for the medium and high complexity cases (shown in Fig. 4-36). Note that the y-axis in this figure is *not* logarithmically scaled. Again, as before, results are only shown for the runs that returned feasible solutions.

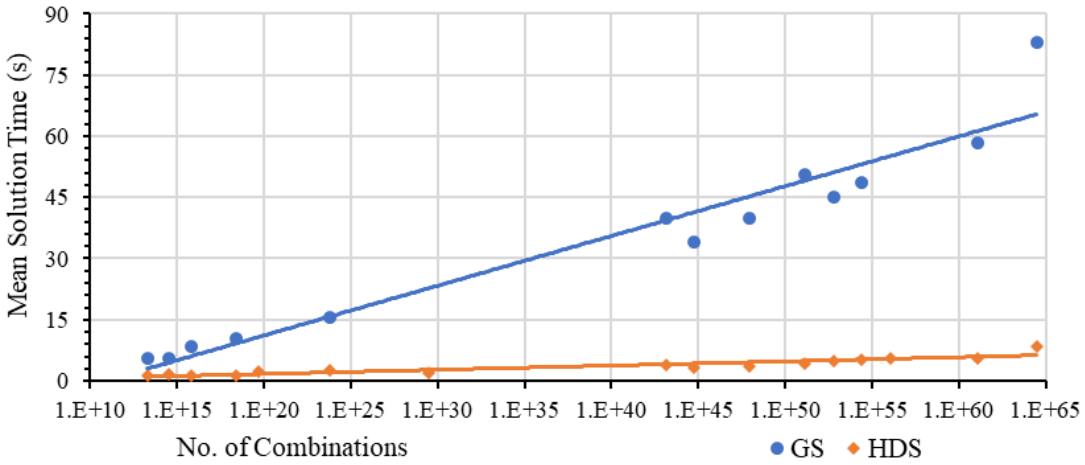


Fig. 4-36. Run times for the GS and HDS algorithms plotted against the number of disassembly sequence combinations for the medium and high complexity, free-rotating cuts scenario. Image adapted from [180].

From this figure, it can be seen that even for the highest number of combinations, all the algorithm search times remain under 90 seconds (which is considerably faster than the 3 hours for all the STS solutions for the same complexity cases). It is also more clear from this figure that, whilst the run times for both the HDS and GS algorithm scale linearly, the run time for the GS algorithm shows a steeper increase in the gradient for increasing complexity.

This is due to the fact that for the greedy search, each time it has to select a part, it must test every remaining part in the structure (for the 3 constraints), before selecting the part with the shortest trajectory. It therefore follows that the computational complexity will scale linearly with an increase in the number of parts. In contrast, for the HDS algorithm, since it always seeks to remove the highest feasible part in the structure, it does not require testing every remaining part in the structure at each step. Instead, it tests the remaining parts in height decreasing order, stopping at the first feasible part that can be removed. This greatly reduces the computational complexity when compared to the GS algorithm (by reducing the number of constraint checks required per iteration), meaning the linear increase in computation time is much less dramatic.

As a final point to note regarding run time, only the results for the free-rotating cuts scenario are presented as the same computation trends were observed for the orthogonal cutting cases as well. Hence these results are omitted for brevity.

Having examined the consistency of each algorithm in producing optimal and feasible solutions, as well as each algorithm's computational efficiency, for the final section of the results analysis, attention is turned toward the quality of the solutions produced (i.e. how well each algorithm performs in terms of minimising the total disassembly cost). For comparing results across the simulations, a metric (which was devised by the student)

called ‘cost strength’ is used, which assesses the relative strength of each algorithms solution for a single cutting pattern. This metric evaluates each algorithm’s ability to approach the optimum solution (where ‘optimum’ refers to the best solution found for a cutting pattern), by comparing its returned cost to the best (minimum) observed cost for that cutting pattern. Specifically, the cost strength is calculated as:

$$Cost\ Strength = \frac{Min\ Returned\ Cost}{Algorithm\ Returned\ Cost} \tag{7}$$

This equation gives a fractional value between 0 and 1 where:

- 1 indicates that the algorithm returned the lowest (i.e. optimum) cost.
- Less than 1 indicates a deviation from the optimum cost (with lower values signifying a greater deviation, and hence poorer performance).
- 0 indicates an infeasible solution.

As a simple example, consider the 4 cases presented below:

$$FFRS\ cost\ strength = \frac{88}{92} = 0.957 \qquad HDS\ cost\ strength = \frac{88}{94} = 0.936$$

$$GS\ cost\ strength = \frac{88}{0} = 0 \qquad STS\ cost\ strength = \frac{88}{88} = 1$$

In this example, the STS algorithm returned the lowest cost (resulting in a cost strength of 1), and the GS algorithm failed to find a solution (resulting in a cost strength of 0). The HDS and FFRS algorithms returned suboptimal solutions (with higher cost than the best) and hence have fractional values between 0-1.

For each cutting pattern tested, the cost strength of the four algorithms used to evaluate it is computed. For each complexity level (low, medium and high) the average cost strength across all the cutting patterns for each algorithm is calculated. Finally, for each algorithm, the average cost strength across all problem complexities is calculated. These results are listed in Table 4-7.

TABLE 4-7. Average cost strength across different problem complexities and cutting scenarios.

	Free Rotating Cuts				Orthogonal Cuts			
Complexity Level	FFRS	GS	HDS	STS	FFRS	GS	HDS	STS
Low	0.985	0.875	0.963	1.000	0.960	1.000	0.941	1.000
Medium	0.140	0.710	0.898	1.000	0.137	1.000	0.662	1.000
High	0.000	0.888	0.869	0.997	0.000	0.875	0.754	0.987
Avg.	**0.375**	**0.824**	**0.910**	**0.999**	**0.366**	**0.958**	**0.786**	**0.996**

Based on the results presented in Table 4-7, it is evident that for both cutting scenarios (free-rotating and orthogonal), the STS algorithm has the highest average cost strength, with the average across all complexities (for both scenarios) being close to 1. As before, it is noted that the only cases where the STS fails to achieve a perfect score (i.e. the cases where the STS algorithm failed to find the best solution) is for the high complexity cases. However, given that the average scores for the high complexity cases are very close to 1, this suggests that even when the STS algorithm finds suboptimal solutions, they tend to be very close to the optimum. In contrast, it can also be seen that the FFRS algorithm performs the worst overall, with very low averages overall for both complexities (and with the averages decreasing greatly as the complexity increases).

Regarding the GS algorithm, the results in Table 4-7 confirm the observation that the GS algorithm performs better on average for the orthogonal cuts when compared to the free-rotating cuts (with the average cost strength across all complexities being higher for the orthogonal cutting scenario compared to the free-rotating cuts scenario). The reason for the poorer GS performance for the free-rotating cuts is due to the multiple failures encountered (which drags the average scores down), whereas it only failed once over all the complexities for the orthogonal cutting. This observation aligns with the previous observation that the stability constraint is more likely to be violated for the free rotating cuts (due to the production of more irregular cut parts). This reinforces the hypothesis that orthogonal cutting leads to a reduction in the risk of instability during disassembly, thereby allowing the GS algorithm to perform more reliably (and effectively).

In contrast, the opposite appears to be true for the HDS algorithm, with the average cost strength across all complexities being higher for the free-rotating cuts when compared to the orthogonal cuts. This is due to the fact that HDS prioritises top-down deconstruction which means it is more likely to find feasible solutions which do not violate the stability constraint. Hence, it has a higher average score for the free-rotating cuts since the GS algorithm failed multiple times (which lowers the average), whereas HDS did not fail at all. In contrast, the GS algorithm only failed once across all complexities for the orthogonal cutting scenario, meaning the average cost strength is less affected by outlier failures. Given that the average cost strength across all complexities for orthogonal cutting is higher for the GS algorithm compared to HDS, this suggests that when the risk of violating the stability constraint is reduced (as is the case with orthogonal cutting), GS can find solutions which are closer to the optimum. As such, it can be concluded that while HDS is more

reliable in maintaining feasibility across the varying conditions, GS tends to produce much higher quality solutions when it does succeed.

4.4.5 Concluding Remarks: Disassembly Sequencing for Cut Structures

Based on the comparative study conducted between the 4 proposed algorithms, their performances can be summarised as follows:

- **FFRS** – This algorithm showed strong performance for the low complexity cases (i.e. cases with few cut parts), managing to find good solutions in a short time for both orthogonal and free-rotating cuts. However, the performance (in terms of feasibility and quality of solutions) rapidly deteriorates for the higher complexity cases, making this approach the worst in terms of problem scalability.
- **STS** – In contrast, this algorithm performed the best overall, managing to always find feasible solutions (even with increasing problem complexity), and managing to consistently find optimal, or near optimal, results. The main downside however is that the computational time scales factorially with an increase in the number of cut parts, with the cut-off time of 3 hours consistently being reached for the medium and high complexity cases.
- **HDS** – This algorithm performed very well in terms of its ability to find feasible solutions (with only one failure case across both cutting scenarios) in a very short amount of time (with the run times for even the highest complexity cases not exceeding 7 seconds). Its good ability to find feasible solutions can be attributed to the top-down disassembly method which leads to a smaller chance of stability constraint violation. The downside however is that since it does not directly optimise the cost objective, it tends to produce highly suboptimal results.
- **GS** – This algorithm showed strong performance in terms of the quality of solutions it finds, with most successfully found solutions being optimal. It also performs well from a computational perspective, with the run times not exceeding 90 seconds even for the highest complexity cases. The main downside however is that it has a tendency to get stuck in infeasible optima, especially when the cut parts are more irregular (as is the case with parts produced from the free-rotating cuts).

Future Work

Unfortunately, due to time constraints, it was not possible to integrate the disassembly sequencing approach into the 3D cutting and packing algorithm. However, before such integration can be achieved, there are several important factors which must be addressed:

Implementing a fast and robust optimisation algorithm: Given that the STS algorithm produced feasible and optimal/near optimal solutions, this algorithm would appear a natural choice for integration. The problem however is the run time which scales factorially with the number of cut parts. Given that the goal of this disassembly process is to integrate it into the cutting and packing algorithm to assess whether cutting patterns can be feasibly disassembled, having to run a computationally expensive disassembly process on each cutting pattern in the total cost GA would likely introduce excessive computational overhead (similar to the performance bottleneck observed with the hyper-heuristic packing algorithm).

Given the success of the GS algorithm in finding optimal solutions in a short space of time, this approach would be a stronger option. However, its tendency to get stuck in infeasible optima poses a challenge. As such, a novel hybrid algorithm that combines elements of GS, HDS and STS is proposed.

The proposed method retains the greedy, deterministic search approach of the GS algorithm, but with a height-based backtracking mechanism. Unlike the purely stochastic backtracking used in the STS algorithm, this approach aims to leverage information to guide the backtracking process. Specifically, the algorithm will track the height of each part as it is removed from the structure. If the greedy search process leads to an infeasible dead end, it will backtrack to an earlier decision point where a lower part of the structure was removed early in the sequence, and explore the alternative branch which delays the removal of that low part until later in the search. This heuristic-driven backtracking approach is based on the observation that removing lower components from the structure early in the search process increases the likelihood of instability later in the search. By using this heuristic backtracking approach, the aim is to escape dead ends efficiently (using height information to guide the recovery) without the computational cost of full stochastic sampling.

Evaluating the accuracy of the stability model: The current implementation used to assess stability is a simplistic model aimed at reducing computational overhead (when compared, for example, to more accurate force-based simulation approaches like Finite

Element Analysis). Given that the goal is to use this algorithm to assess all the cutting patterns produced by the cutting and packing GA, it is highly desirable to keep the computation time of this sub-process to a minimum.

Future work should focus on evaluating how well the simplified stability model approximates real structural behaviour by benchmarking it against more advanced physics-based models. If the current approach is found to be sufficient at modelling instability, it can be retained for the 3D algorithm. If it is found to be unsuitable, a hybrid approach could offer an effective compromise.

The idea behind the hybrid approach is that the existing stability model would act as a primary estimation process. Then, for cases where the structural stability is only marginally violated (such as with the example in Fig. 4-34), a more accurate physics-based method could be triggered. This would allow for improved accuracy in cases where instability is more uncertain without adding significant computational overhead.

Robotic manipulator planning: A key goal moving forward is to transition from purely algorithmic disassembly sequencing to fully executable robotic disassembly. To achieve this, several challenges relating to robotic manipulator planning must be addressed.

Firstly, whilst the current algorithm is able to generate sequences that satisfy the collision free trajectory constraint, the collision free requirement only applies to the part itself, i.e. there is no consideration for the kinematic of physical limitations of a robotic manipulator system. In real world robotic applications, robotic manipulators must be able to physically reach each part to remove, must be able to grasp it, and must execute the removal trajectory without self-collisions or joint limit violations. As such, future work must also focus on integrating more sophisticated motion planning algorithms that can accurately account for a robot's degrees of freedom, reachability in the workspace, and environmental constraints (such as obstacles the manipulator must navigate around).

Secondly, it would also be desirable to allow the parts to follow non-linear trajectories. Whilst enforcing linear trajectories simplifies the problem from a computational perspective, it significantly limits the flexibility of the algorithm (e.g. a part which cannot be removed under a linear trajectory may be removable using a non-linear one). Therefore, future work should also focus on implementing more sophisticated non-linear path planning algorithms for the part trajectories.

Finally, cutting and grasp planning should also be incorporated to check if a.) a robotic manipulator fitted with a cutting tool can physically perform the cutting action (i.e. that the

robot can follow the cutting line with its tool without any part of the robot colliding with parts of the structure) and b.) a robotic manipulator fitted with a gripper can grasp the part without it slipping and dropping. Therefore, any future work which aims to incorporate kinematic models for the robotic manipulators must also account for this.

A point worth noting is that implementing the aforementioned robotic capabilities would likely introduce large computational overhead. As such, future work should focus on developing fast approximation approaches that can predict robotic feasibility without having to perform full scale dynamic simulations. If such approaches prove to be insufficiently accurate, then a two-stage approach could offer a potential solution. In such an implementation, the fast model would be used within the cutting and packing algorithm (to retain computational efficiency), with a more advanced dynamic model being run as a post processing step to evaluate the final optimised cutting pattern.

The main challenge here would be to minimise discrepancies between the fast approximation model and the advanced dynamic model. For example, if the post-processing model finds that a cutting pattern is not feasible to disassemble, a fallback strategy would need to be implemented. The idea of the fallback strategy would be to identify the point of failure (i.e. the features of the cutting pattern which lead to infeasibility) and then try to correct them using a repair mechanism to restore feasible disassembly.

How to integrate the disassembly sequence algorithm into the cutting and packing GA: As stated, the future aim with this disassembly algorithm is to integrate it into the cutting and packing optimisation framework. The goal of this would be run the disassembly sequencing process on each cutting pattern produced by the GA to check if it is feasible for disassembly.

One possible option for integration would be to incorporate the disassembly check into cutting cost function, with a penalty term used to degrade the cost based on how ‘infeasible’ a cutting pattern is in terms of disassembly (e.g. based on how much a cutting pattern violates the stability constraint). In doing so, the algorithm would be motivated to favour cutting patterns with feasible disassembly sequences as this would drop the penalty term to 0, lowering the cutting cost. Whilst this penalty-based strategy presents the easiest option in terms of integration into the cutting and packing GA, the main downside is that penalty terms do not guarantee feasibility (i.e. there is no guarantee that the penalty term will be reduced to zero). As such, it is possible that the algorithm may not be able to find a cutting/packing solution where the cutting pattern is fully feasible for disassembly.

An alternative strategy would be to use a repair mechanism to try and enforce feasible disassembly. Rather than simply discarding cutting patterns with infeasible disassembly sequences or penalising them in the cost function, the idea behind a repair mechanism would be to modify the existing cutting pattern to enable feasible disassembly. This could, for example, include modifying cuts to move the projected centre of mass points into the support polygon lines when a cut leads to large unstable overhangs. The main problem with this approach is that it would be challenging to design a fast and efficient repair mechanism which can repair infeasible disassembly sequences without degrading the other optimisation objectives (i.e. the cutting cost and the packing cost). Future work should focus on first testing a penalty-based integration approach (to assess how consistently the algorithm can remove the penalty for infeasible disassembly cutting patterns), and if the approach is found to be unsuitable, exploring the concept of repair mechanisms in greater depth.

4.5 Chapter 4 Summary

In this chapter, the efficacy of the proposed optimisation approach for linking cutting and packing was evaluated. A simple version of the GA-based cutting and packing algorithm was implemented in 2D and benchmarked against a random search. It was found that the GA algorithm outperformed the random search for every trial tested, however some problems with the initial methodology were also noted. In particular, the number of variables required to represent cuts in the cutting method (BSP) coupled with the use of penalty functions for dealing with infeasible solutions resulted in slow convergence times. As such, for the 3D implementation of the algorithm, the cutting approach will be simplified (to orthogonal cutting) to reduce the number of optimisation variables, and to allow the implementation of a simple repair mechanism for infeasible cutting patterns (thereby removing the need for penalty functions).

In addition to this, several projects were carried out to investigate ways to improve the cutting and packing processes within the algorithm. These were:

Hyper-heuristics for container packing: To improve the quality of solutions produced by the DigiPac packing algorithm, a novel hyper-heuristic approach was implemented, both for single container packing (where the choice of placement heuristic for packing objects was optimised), and multi container packing (where both the placement heuristics and packing order was optimised to reduce the number of containers required to pack objects). The single container hyper-heuristic was compared against a conventional single-heuristic

order optimisation approach, with the proposed hyper-heuristic shown as being able to outperform it in every trial. The methodology was then extended to multi container packing and benchmarked against literature. It was found that the algorithm was able to outperform literature for datasets containing a small number of objects but struggled to find good solutions for datasets with a large number of objects. Based on the findings, a novel hyper-heuristic framework was proposed, which aims to leverage information about the properties of both the shape to pack and the existing packing pile to intelligently guided the choice of placement heuristics.

Allocation-based multi-container packing: Based on the strength of the allocation-based packing approach, which was benchmarked against in the hyper-heuristic study, the decision was made to implement their multi-container strategy with the DigiPac placement approach. The algorithm was then tested on multiple datasets (of increasing problem complexity) for each of the placement heuristics. From this study it was shown that the allocation-based strategy performs much faster than the hyper-heuristic algorithm and can consistently produce higher quality solutions for the datasets with many objects. It was also noted that, of the 4 placement heuristics tested, height minimisation and contact area maximisation performed the best on average. Based on these findings, the decision was made to use the allocation-based packing approach for the 3D cutting and packing algorithm, with height minimisation as the choice of placement heuristic.

Disassembly sequencing for cut structures: In this study, a novel methodology for calculating mechanically stable and physically realisable disassembly sequences for pre-partitioned structures was proposed. 3 different optimisation algorithms for optimising the disassembly sequencing were presented and compared: a greedy approach, a heuristic driven approach and a tree-search approach. From the comparative study it was found that the tree-search algorithm performed the best overall in terms of its ability to find feasible disassembly sequences, however it was also noted that the computation times scale factorially with the problem complexity. In contrast, the greedy search performed well in terms of finding optimal solutions in a short space of time, however it also had a greater tendency to get stuck in infeasible optima. The heuristic driven approach also performed very well in terms of finding feasible solutions in a short space of time, however the solutions found were often suboptimal. Based on the findings, a novel algorithm is proposed, which aims to combine the speed and efficacy of the greedy search algorithm with the backtracking ability of the tree-search algorithm to help it escape infeasible local optima. For the backtracking process, a new heuristic driven approach is proposed, which aims to backtrack to points in the search which likely lead to constraint violations.

Additionally, several improvements are discussed for future work, including more advanced stability calculation techniques as well as the incorporation of robotic models for accurately modelling the removal process for cut parts.

CHAPTER 5
3D WORK

This chapter presents the implementation and testing of the 3D version of the cutting and packing algorithm developed in this study. Building on the methodology proposed in Chapter 3 and the 2D work in Chapter 4, this chapter extends the approach to handle more realistic 3D structures and explores the effectiveness of the algorithm under more complex and practical conditions.

The primary objective of this chapter is to evaluate the performance, flexibility and robustness of the 3D algorithm across a variety of decommissioning scenarios. This includes investigating how well the algorithm can minimise total cost, how different parameters affect solution quality, and how the algorithm compares against alternative approaches including manual methods.

The remainder of this chapter is structured as follows:

- **Section 5.1 – Updated Methodology for 3D**: Outlines the modifications made to the original methodology presented in Chapter 3, based on the insights gained from the 2D studies presented in Chapter 4. The rationale behind these changes is also discussed.
- **Section 5.2 – Implementation:** Provides details on the algorithm's implementation in 3D, with focus on how the cutting and packing components were implemented, and how the initial population was generated.
- **Section 5.3 – Hyperparameter Tuning:** Presents a study into the effects of key genetic algorithm hyperparameters on optimisation performance. The outcomes from this study are used to set the hyperparameters for the remainder of the chapter.
- **Section 5.4 – Simulation Setup:** Describes the test scenarios used to evaluate the algorithm, including three hypothetical decommissioning cases, benchmarking against a manual method and cost-weight sensitivity analysis.
- **Section 5.5 – Results:** Presents the results from the tests described in Section 5.4, including a detailed analysis of algorithm convergence behaviour, cutting and packing performance across different structures, benchmarking against manual approaches, and the effects of varying cost-weight parameters on solution quality.

5.1 Updated Methodology for 3D

This section outlines the updated cutting and packing methodology developed for the 3D implementation of the algorithm. As with the 2D version (outlined in Chapter 3, Section 3.2.2), the approach for solving cutting and packing is a feedback-driven optimisation loop in which a genetic algorithm (GA) generates candidate cutting patterns which are then evaluated based on both the cutting cost and packing cost. The cutting algorithm segments the input object according to each candidate pattern (and calculates the cutting cost of each pattern), and the resulting parts are then passed to a packing algorithm to pack them (and calculate packing costs). These costs are then fed back into the genetic algorithm to guide the generation of improved cutting patterns.

One key change to this methodology in the 3D version is that the GA no longer uses a pareto-based ranking system. Instead, a weighted cost function (explained in Section 5.1.3 below) is used to sum the individual costs to give a single scalar value (i.e. total cost) for each cutting/packing solution. This allows the GA to simply rank the solutions from best to worst (i.e. lowest total cost to highest), based on this scalar value. The updated feedback structure is illustrated in Fig. 5-1, where the cutting and packing costs are now explicitly summed into total cost values before being used by the GA for selection and reproduction.

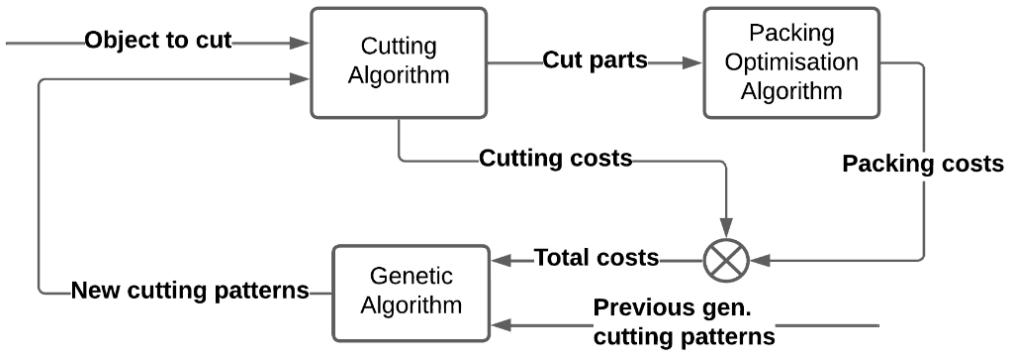


Fig. 5-1. Flowchart outlining the updated cutting and packing optimisation process.

In addition to this change, another key change is the use of voxel representation throughout the 3D algorithm (i.e. for both cutting and packing), rather than a dual representation approach (mesh for cutting, voxels for packing) as used in the 2D version. As stated in Chapter 3 (Section 3.1.1), the reason for this change is that it removed the need for repeated format conversions between the cutting and packing processes. Instead, the model only needs to be converted to voxels once at the start, reducing computational overhead.

In addition to these changes, several other modifications have been made to the methodology based on findings from the 2D testing work (Chapter 4). These include updates to the cutting strategy (Section 5.1.1), updates to the packing strategy (Section 5.1.2) and updates to how cost is represented (Section 5.1.3).

5.1.1 Updated Cutting Approach

The 2D implementation of the cutting and packing algorithm revealed several problems with the chosen cutting strategy and the way infeasible solutions were handled. Initially, the algorithm used a Binary Space Partitioning (BSP) approach (outlined in Chapter 3, Section 3.1.2), which recursively divides the object using planar cuts. Infeasible solutions (i.e. cutting patterns which produced cut parts larger than the container dimensions) were penalised using a penalty approach to degrade the cutting and packing costs.

Based on the results presented in Chapter 4 (Section 4.1.3) it was noted that the problem with both choices is that it resulted in slow convergence behaviour, with the algorithm requiring long computation times to obtain even feasible solutions. It was theorised that the source of the problem was likely two-fold; a high number of optimisation variables arising from the fact that each cut required 3 variables (in 2D) to represent it (a centre point given as x, y coordinates and a rotation about the centre point given by θ), and the use of penalty functions which resulted in the algorithm spending unnecessary additional time evaluating infeasible solutions.

Based on these observations, the fallback cutting strategy (Chapter 3, Section 3.1.2) of orthogonal planar cutting was adopted for 3D, where each cut is orthogonal to the x, y or z axis of the structures bounding box and is allowed to pass through other cuts (Fig. 5-2).

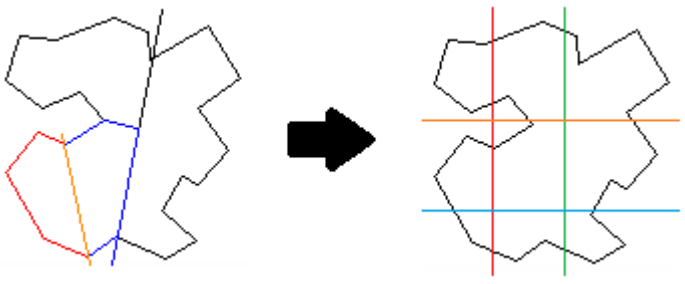


Fig. 5-2. Original BSP cutting approach (left). New orthogonal cutting approach (right). Images adapted from [130].

This updated methodology offers several advantages:

- Each cut now requires only one variable (to define its position along a fixed axis), significantly reducing the number of optimisation parameters. In contrast, BSP in 3D would require six variables per cut ($x, y, z, \theta_x, \theta_y, \theta_z$), which could make the problem intractable for large structures.
- The simplified representation enabled the introduction of a simple repair mechanism, which inserts additional cuts if the gaps between existing cuts exceed the container size limits (see Section 5.2.1). This ensures all generated cutting patterns are feasible.
- Penalty functions are no longer required (as the GA now operates solely within the feasible search space) helping to improve the search efficiency.

As an additional point to note, whilst the disassembly sequencing algorithm (Chapter 4, Section 4.4) couldn't be implemented into the 3D algorithm due to time constraints, one of the key observations from this study was that orthogonal cutting tends to facilitate easier disassembly. It was noted that orthogonal cuts tend to produce parts which are more 'regular' (less variation is size and shape) which lowers the risk of instability during the disassembly process. Given that the goal in future is to integrate the disassembly sequencing algorithm into the 3D cutting and packing algorithm, the benefit of using orthogonal cuts is that it should facilitate easier disassembly, allowing the sequencing algorithm to find optimum disassembly sequences faster and with a reduced chance of failure.

5.1.2 Updated Packing Approach

In the 2D work, a multi-container hyper-heuristic packing strategy was developed which combined packing order optimisation with placement heuristic optimisation (Chapter 4, Section 4.2.3). Whilst the approach was shown to perform well on datasets with a small number of objects, it struggled with scalability due to the factorial growth in the number of permutations as the number of objects increased, leading to slow convergence and inconsistent performance.

Given that real-world decommissioning can involve packing hundreds of irregular cut parts, and that each modification to a cutting pattern changes the object set (requiring the re-packing of the new parts), this approach was deemed impractical for the 3D

implementation. A faster and more scalable packing strategy was required, one which could pack large object sets in a fast and consistent manner.

To address this, the Partial Bin Packing (PBP) strategy was adopted for the 3D implementation. As demonstrated in the 2D comparative study (Chapter 4, Section 4.3), compared to the hyper-heuristic algorithm, PBP achieved strong performance with significantly reduced computation time by decoupling object allocation from the packing order. PBP works incrementally, packing one container at a time and selecting subsets of objects using a knapsack-based allocation strategy. This avoids the need for repeated full-set repacking (as is the case with order optimisation), making it particularly well suited for large scale problems with many objects.

Additionally, among the placement heuristics tested with PBP in 2D, it was found that height minimisation produced the best overall packing efficiencies. As such, this placement heuristic was selected for the 3D implementation. In addition to its efficiency, height-based approaches have also been shown in literature to be effective at promoting stability in the packing pile [55]. Although physical stability is not explicitly modelled in the current 3D implementation, this is a constraint which will be considered in future work. Using height minimisation therefore presents a strong choice of placement heuristic as it is expected to encourage more inherently stable packing arrangements.

As a final point to note, for the 3D implementation, an ‘accessibility’ constraint was also added, to ensure that all objects can be placed in the desired location without colliding with previously packed objects or the container boundary. In the 2D work, accessibility was not considered, meaning that smaller objects would often be placed in gaps underneath previously packed objects. Whilst this may not be problematic for 2D packing problems (where the goal is often to pack a set of shapes onto a sheet of material before cutting them out), the issue with 3D packing is that objects cannot be placed underneath packed objects without collision. As such, accessibility was necessary to ensure that objects will only be placed in locations which can be reached from the top of the container. Although this constraint may reduce the overall packing efficiency by limiting the number of placeable locations (see example in Fig. 5-3), is necessary for maintaining real-world feasibility.

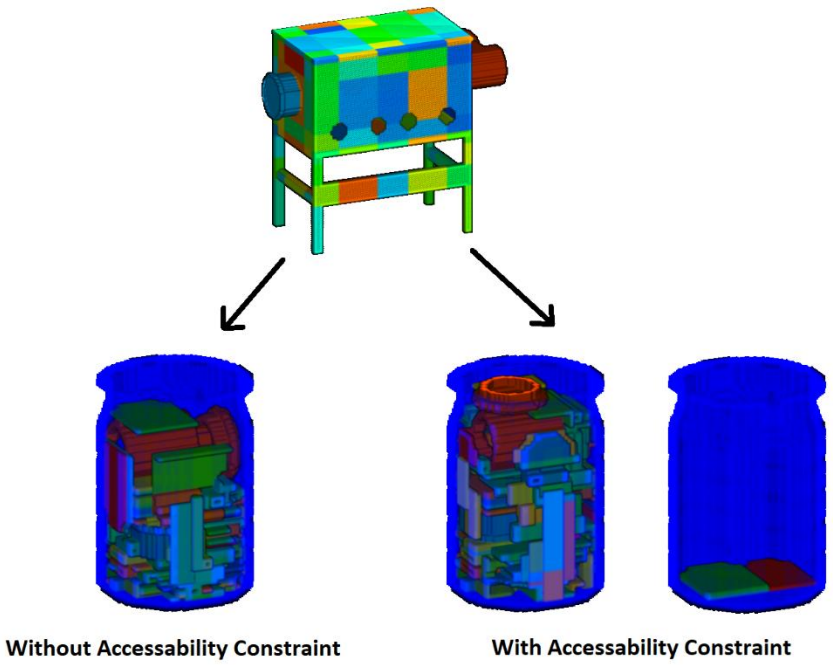


Fig. 5-3. Example of a cut structure packed with and without accessibility constraints. Without accessibility constraint, all parts can fit into a single container.

5.1.3 Updated Representation of Total Cost

In the 2D implementation of the cutting and packing algorithm, the cutting and packing cost was optimised using a separate pareto-based optimisation framework. For the 3D algorithm, the two cost functions were instead combined into a weighted objective function, with the cost weights representing problem specific weights relating to the unit-cost of cutting and packing. The decision to adopt a weighted objective function for 3D was made for several reasons:

- **Improved interpretability:** Using a single scalar cost function gives a clear and intuitive indication of overall performance (i.e. the success of the algorithm can be directly measured in terms of cost decrease) and also allows the user to easily identify the cheapest solution without requiring any subjective decision making (such as selecting from a set of pareto optimal solutions).
- **Simplification of the decision making:** The problem with having a set of pareto optimal solutions is that, for an algorithmic perspective, all of them are considered equally optimal. Whilst this may be beneficial in terms of providing users with a set of options, the problem is that it does not help the user make the final selection. Given that the goal in cutting and packing is to select the most cost-effective solution, users would need a way to convert the pareto optimal solutions into single cost values to allow for the selection of the cheapest solution. The problem here is

that this makes the multi-objective optimisation framework redundant. I.e. if the user needs a cost function to convert pareto-optimal solutions into cost values, a simpler approach would be to input the function directly into the algorithm.

- **Framework flexibility:** Another benefit of using a weighted cost function is that it makes the algorithm more flexible in terms of modification for specific problems. If a specific decommissioning scenario requires additional cost components, these can easily be integrated into the cost function (either by modifying the existing terms or adding new terms) with no modification of the existing optimisation logic being required.

The updated method for calculating total cost is given as follows. As with the 2D implementation, the cutting cost is defined as the intersection volume between the cuts and the structure (V_c). This value is then multiplied by a cost weight (w_1), which represents the unit cost per voxel of cutting. The unit cost per voxel (w_1) is a problem specific parameter which is calculated by the user, and would need to consider factors such as the operational cost of the cutting tools as well as any associated labour costs, cost of using teleoperated robots (if applicable), maintenance/repair costs, etc. Additionally, since the user can define the width of the cuts, if the user selects zero-width cutting, the intersection volume is instead calculated as the intersection area between the cuts and the structure (see Fig. 5-4).

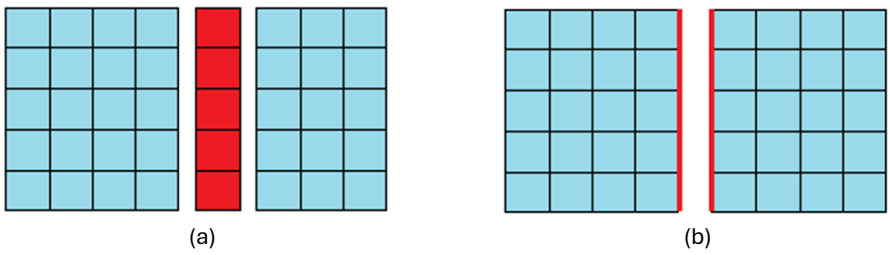


Fig. 5-4. 2D illustration of different cut widths. (a) cut width = 1, red region is the intersection volume of the cutting plane and the object. (b) cut width = 0, red lines are the intersection area between the cutting plane and the object.

Since the 3D implementation of the algorithm has multi-container packing integrated, rather than using the percentage space utilisation of a single container (as with the 2D implementation), the packing cost is now calculated based on the number of containers (N_p) required to pack all the cut parts. As with cutting, the number of containers is multiplied by a cost weight (w_2), which represents the unit cost per container. Again, this problem specific parameter would be calculated by the user and include factors such as

the manufacturing cost of the container, transportation and storage costs, and any associated labour/equipment costs.

Thus, the total cost for a cutting and packing solution can now be expressed as the weighted objective function:

$$T_{cost} = w_1 V_c + w_2 N_p \quad (8)$$

Where the goal is to minimise T_{cost}, subject to the following constraints:

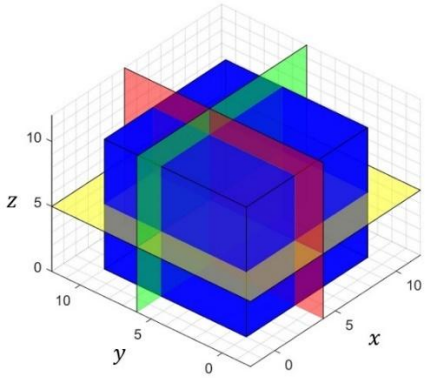
1) All cut parts must be small enough to fit into the containers.
2) There must be no overlap between packed parts and all packed parts must be within the container boundaries.
3) Each packing location must be accessible, with a linear collision-free trajectory from the top of the container to the desired packing location.

5.2 Implementation

The algorithm is implemented into the decommissioning software NuPlant (Developed by StructureVision Ltd. at the University of Leeds [127]), written in C++, and was run on a desktop PC with an Intel Xeon E5-1650 v3 3.50GHz processor and 64Gb of RAM. In NuPlant, the input structure is converted to voxel representation, and the packing of the cut parts is done using the DigiPac packing algorithm (also developed at the University of Leeds [47]).

5.2.1 Orthogonal Cutting

Referring to Fig. 5-5 as an example, a list of cuts is implemented as a 3-column vector, where each row corresponds to a single cut, and the position of the non-zero entry in each row determines which axis the cut is orthogonal to. The value assigned to the non-zero entry in each row dictates where along the axis each cut lies with respect to the origin point (front-bottom-right of the structures bounding box).



X	Y	Z
4	0	0
0	6	0
0	0	5

Fig. 5-5. Illustration of orthogonal cutting in 3D. Left - blue cube cut with three planar cuts orthogonal to: x axis (red), y axis (green) and z axis (yellow). Right - cutting list with corresponding cuts colour coded.

As with the 2D implementation, when performing crossover on two cut lists, single point crossover is used where both cut lists from both parents are split at random points. The top and bottom halves of the lists from both parents are then recombined to form two offspring. For mutating offspring, point mutation is used where the algorithm will cycle through a list of cuts, with each cut having a chance of being mutated (where the chance of mutation is dependent on the user-set mutation rate). When a cut is mutated, the algorithm will apply a small random change to the position of the cut to move it in the positive or negative direction.

For the repair mechanism, at the algorithms start, the user is asked to set maximum x, y, z limits for the cut parts (which must be smaller than or equal to the internal container dimensions). If the algorithm finds that the distance between two consecutive cuts along an axis exceeds the limit set for that axis, the distance between the two cuts will be recursively divided until all sub-sections are smaller than the limit. The repair mechanism also removes duplicate cuts (cuts on the same axis with the same values) and cuts that have been mutated outside the structures bounding box.

5.2.2 Partial Bin Packing

The process for PBP is explained as follows (for a more detailed outline the reader is referred to Chapter 4, Section 4.3.1). The packing algorithm starts by opening an empty container and running an allocation algorithm to select a subset of cut parts to allocate to the container. The allocation algorithm itself is a modified version of the knapsack problem where objects are selected to maximise the sum of their volumes without exceeding the available volume in the container. To reduce the greediness of the allocation algorithm, a modified objective function is used which favours assigning larger objects. This aims to

prevent greedy (and poorer) solutions where small objects are packed together early on, leaving larger objects till the end [178].

The subset is then ordered by decreasing volume, and the parts are packed one by one, starting with the largest. If the algorithm finds that an object won't fit, the allocation is mended. During mending, the successfully packed parts are kept in the container and the object which failed to fit is placed into a temporary exclusion set. The allocation process is then re-run to select another subset from the unpacked items to try adding to the partially packed container. This process of packing and mending the assignment will repeat until either all allocated objects are successfully packed, or there are no more objects left in the main object set to try adding to the container. Upon termination the packed container is closed, any objects in the temporary exclusion set are returned to the main object pool, a new container is opened, and the process repeats. This process, along with the repair loop, is illustrated in Fig. 5-6.

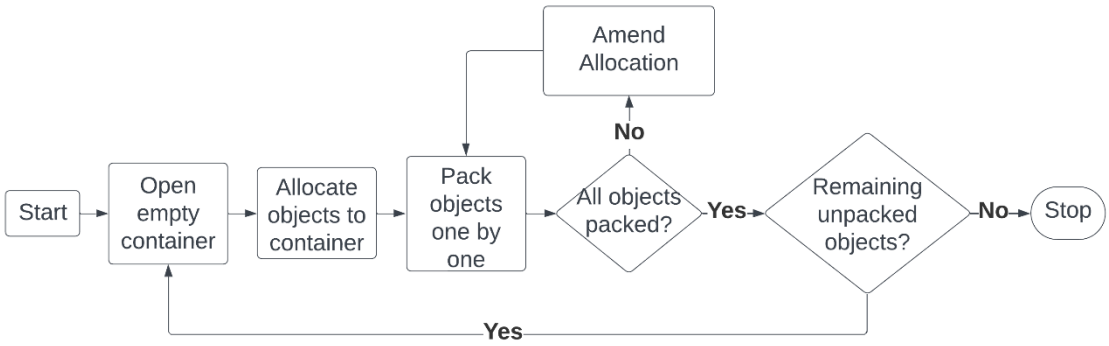


Fig. 5-6. Simplified illustration of PBP approach.

When determining where to place an object in the container, as with the 2D implementation, the DigiPac packing algorithm is used, where the voxelised object is translated, in discrete steps, across the discretised packing space for all allowed orientations (with the angle increment set by the user). At each step, overlap detection is performed to check for collisions with previously packed objects or the container boundary. If there is no overlap, the location is stored as a feasible packing site. The object is then placed at the location which minimises its height in the packing structure and is checked for accessibility by translating it vertically to the top of the container, whilst checking for overlap with previously packed objects and the container boundary. If no overlap occurs, the object is placed. If there is overlap, the next best site is tested. This process continues until an accessible site is found or the object cannot be packed.

For solving the allocation problem, as with the 2D implementation, a simple genetic algorithm is used. In this GA, the optimisation variable is a bit string of length n (where n is the number of objects to pack), with a value of 1 in position $i, i \in 1, \dots, n$, of the bit string indicating that object i is included in the allocation, and a value of 0 indicating that object i is not included. For crossover, single point crossover is used, where the same random split point is used for both parents (to ensure both child solutions remain at length n). For mutation, random bit flip mutation is used where the algorithm will cycle through a bit string produced via crossover, with each bit having a small probability of being flipped (changed from 0 to 1, or vice versa).

5.2.3 Population Initialisation

To generate the initial population, 12 cutting patterns are generated using fixed minimal-cutting approaches, with the rest of the population being generated randomly. For the fixed cutting patterns, 4 different minimal cutting approaches (outlined in detail below) are used, which are initialised 3 times with different maximum allowed size limits ('small', 'medium' and 'large'), giving 12 cutting patterns in total. Recall from Section 5.2.1 that when the algorithm is initialised, the user is asked to set maximum x, y, z limits for the cut parts. For the 'large' size limit solutions, 4 different minimal-cutting solutions are generated with the maximum allowed part size set to $1 * \{x, y, z\}$, i.e. with the part limits at their full size as set by the user. For the next 4 ('medium' size-limit) cutting patterns, the same 4 approaches are used to generate the minimal-cutting solutions, but with the part size limits set to $0.66 * \{x, y, z\}$. For the final 4 ('small' size-limit) solutions, the process is repeated but with the limits set to $0.33 * \{x, y, z\}$. Fig. 5-7 shows an example of a structure cut using the same minimal cutting approach but with different size limits (1, 0.66, 0.33 * $\{x, y, z\}$).

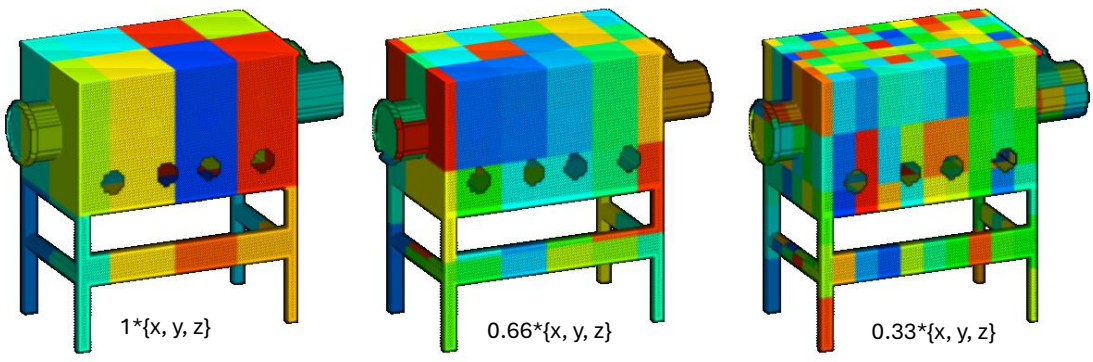


Fig. 5-7. Example of minimal cutting approach with different size limits for the cut parts.

The 4 approaches used to generate the minimal cutting solutions are:

Recursive division - Each axis of the structures bounding box (x, y, z) is recursively split in half until each line segment is smaller than the limit for that axis. Fig. 5-8 illustrates this concept for a single axis of length 9. With an axis limit of 2, recursive division splits the line segment using 7 cuts. The line is first split in half at 4.5, then both halves are split at 2.25 and 6.75, then the remaining 4 segments are split at 1.125, 3.375, 5.625 and 7.875, giving eight segments of length $1.125 < 2$.

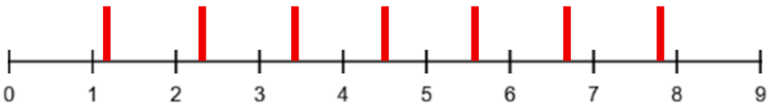


Fig. 5-8. Illustration of recursive division approach. Line segment recursively divided until distance between cuts ≤ 2.

Minimal cutting - For each bounding box axis, calculate minimum number of cuts as $n_c = \lfloor length/limit \rfloor$, then, for $i = 1, \ldots, n_c$, add a cut at $i * limit$. Fig. 5-9 illustrates this with a simple example. With an axis length of 9 and a limit of 2, the minimum number of cuts required to split the line is $\lfloor 9/2 \rfloor = 4$. Cuts are added at 2,4,6,8.

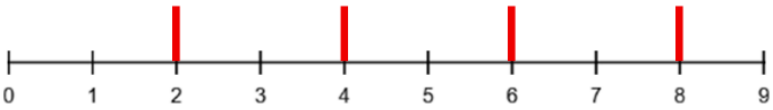


Fig. 5-9. Illustration of minimal cutting approach. Number of cuts to add = ⌊9/2⌋ = 4, cuts added at 2,4,6,8.

Even division - Divide each axis with evenly spaced cuts where the distance between cuts is smaller than its respective axis limit. Fig. 5-10 illustrates this with a simple example. In this case, the line segment is split into 5 equal segments of size 1.8 with 4 evenly spaced cuts.

Fig. 5-10. Illustration of even division approach. Distance between cuts = 1.8.

Random single cut - For each axis, place a single cut at a randomly generated point, then check the length of the line segment either side of the cut. If either one exceeds its

respective axis limit, recursively divide the line segment. In the example shown in Fig. 5-11, a randomly generated cut is added at 4; the line segments either side are then recursively divided until all line segments are within the limit.

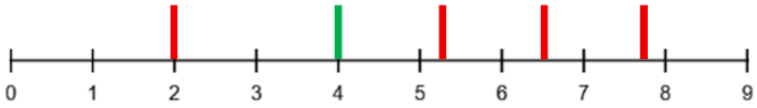


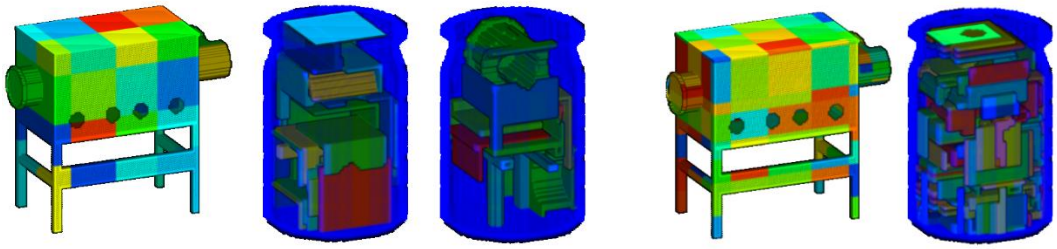
Fig. 5-11. Illustration of random single cut approach. Random cut (green) added at 4, line segments either side recursively divided.

The four minimal cutting approaches outlined above correspond to baseline cutting strategies that were already implemented in the NuPlant software prior to the work conducted in this thesis. Additionally, the choice of using this combined approach (of fixed cutting patterns combined with randomly generated ones) was made for several reasons:

- Firstly, by initialising the algorithm with pre-existing minimal cutting strategies, it creates a baseline for comparison, where solutions produced by the integrated cutting and packing algorithm can be compared to solutions produced by minimal cutting followed by packing optimisation. A major limitation with existing cutting algorithms and packing algorithms developed for nuclear is that they do not consider the trade-off between the two processes, instead optimising them in isolation. i.e. cutting algorithms aim to minimise cutting without considering packability, while packing algorithms work with fixed pre-cut parts. By initialising some of the population members with minimal cutting approaches, the aim is to show how an integrated approach can outperform separate optimisation strategies with minimal cutting followed by packing.

- Additionally, the choice to initialise the population with fixed patterns with the limits set to 1, 0.66 and 0.33 times the maximum x, y, z limits for the cut parts was made to ensure that the population starts with a good distribution of solutions (i.e. solutions with ‘large’, ‘medium’ and ‘small’ cut parts in them). In early testing of the 3D algorithm, the population members were all initialised randomly (same as for the 2D implementation). It was noticed however that this sometimes resulted in premature convergence to suboptimal solutions. For example, when testing the algorithm on the glovebox structure depicted in Fig. 5-12, it was observed that with purely random initialisation, the algorithm would sometimes converge to a sub-optimal two-container solution (Fig. 5-12 a) with a higher total cost than if the

structure was packed into a single container (Fig. 5-12 b). After implementing the 'small', 'medium', and 'large' cutting initialisation approach, the algorithm would always converge to a more optimal, single container solution.

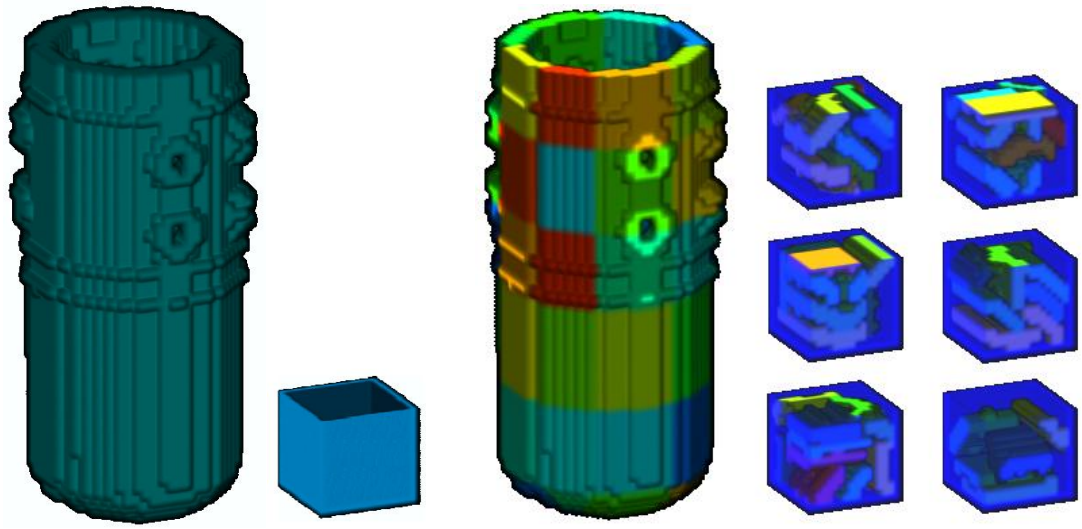


(a) (b)

Fig. 5-12. Optimised cutting and packing solutions for a glovebox packed into drum containers. (a) – purely random initialisation leads to suboptimal solution with two containers. (b) – 'small', 'medium', and 'large' cutting initialisation approach leads to optimal solution with one container.

5.3 Hyperparameter Tuning

Regarding the hyperparameters for the GA, the effects of altering the mutation rate, tournament size (for tournament selection) and the population size were tested to identify optimum parameter choices for the remainder of the tests presented in this study. For testing the parameters, the simple low-resolution model of an RPV (adapted from [203]) was used, with the cut parts being packed into cuboid containers (shown in Fig. 5-13). The voxel dimensions for the RPV (given as l, w, h) are (26, 28, 60) and the dimensions for the container are likewise given as (17, 17, 16). The maximum size limits for the cut parts (given as x, y, z) were set to (15, 15, 15). For all the testing presented in this section, cost weights of $w_1 = 1$ and $w_2 = 100{,}000$ were used, with a maximum number of generations = 100, and an elite population size of 1 used for the GA.



(a) (b)

Fig. 5-13. Structure and container used for hyper-parameter testing. (a) – voxelised input model and container. (b) – example of an optimised cutting and packing solution showing the structure packed into six containers. Note, containers are shown to scale.

5.3.1 Mutation Rate Testing

For testing the effect of mutation rate, 5 mutation rate values were selected to test: 0.1, 0.2, 0.3, 0.4 and 0.5. As a point to note, given the fact that mutation is applied to all child solutions, with the probability of a single cut being mutated determined by the mutation rate, the choice was made not to test values higher than 0.5 since this would cause most of the cuts in child solutions to be mutated. In doing so, any good inherited genes from crossover would likely be lost and the algorithm would degenerate into a purely random search.

For each mutation rate value, the algorithm was run 5 times with the results averaged over the 5 runs. A population size of 20 and a tournament selection size of 0.1 (i.e. 10% of the population size) was used. Fig. 5-14 shows how the total cost of the best solution in each generation (averaged over 5 runs) changes over 100 generations for each mutation rate value tested.

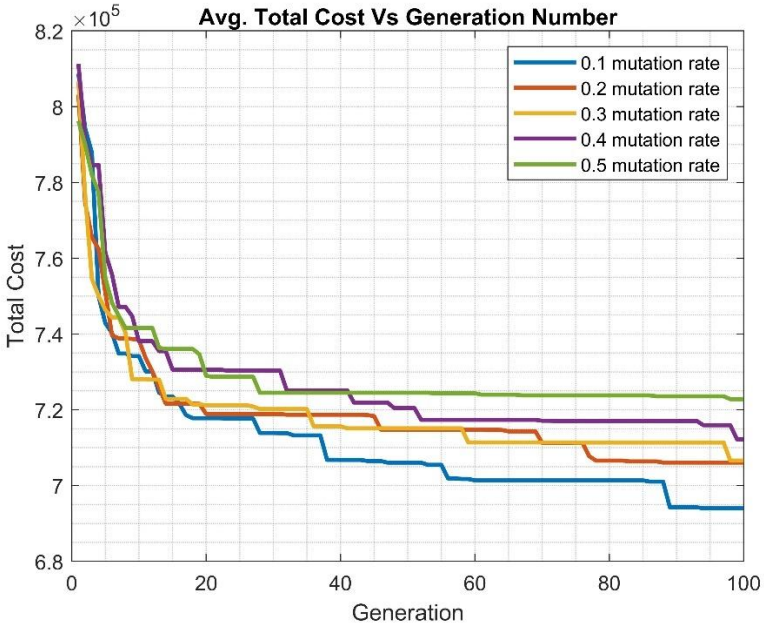


Fig. 5-14. Total cost (averaged over 5 runs) for each mutation rate value tested.

From Fig. 5-14, it can clearly be seen that the runs using a mutation rate of 0.1 gave the best performance on average, with the performance gradually degrading as the mutation rate was increased. This is likely due to that fact that, as the mutation rate increases, it introduces more randomness into the search process, which in turn increases the likelihood of good solutions obtained from crossover being disrupted before they can be refined further. Whilst some mutation is necessary to help maintain diversity and help the algorithm explore the search space better, excessively high mutation rates can prevent the algorithm from converging effectively, leading to premature convergence to suboptimal solutions.

In the 2D implementation of the cutting and packing algorithm (Chapter 4, Section 4.1), the performance of the cutting and packing GA was compared to a random search and it was found that the GA consistently produced solutions of higher quality when compared to the purely random search. This observation coupled with the observation of worse performance as the mutation rate is increased supports the idea that the strength of a genetic algorithm lies in its ability to balance exploration of the search space (through mutation) with exploitation of better solutions to produce new ones (through crossover). The problem with excessively high mutation rates is that it disrupts this balance, causing good solutions produced via crossover to be continuously disrupted by mutations, which ultimately leads to behaviour more closely resembling a purely random search.

This observation becomes more apparent when examining Fig. 5-15 below. This figure shows the average cost values (plotted on separate graphs) with the min/max ranges from the 5 runs indicated by the shaded regions. From these figures, the shaded regions become larger as the mutation rate is increased, indicating that the algorithm loses consistency (i.e. it becomes more prone to premature convergence to poor solutions). In contrast, whilst a slower convergence can be seen on the 0.1 rate graph when compared with the higher mutation rates, it is also evident from the smaller shaded region that the algorithm can more consistently converge to better solutions.

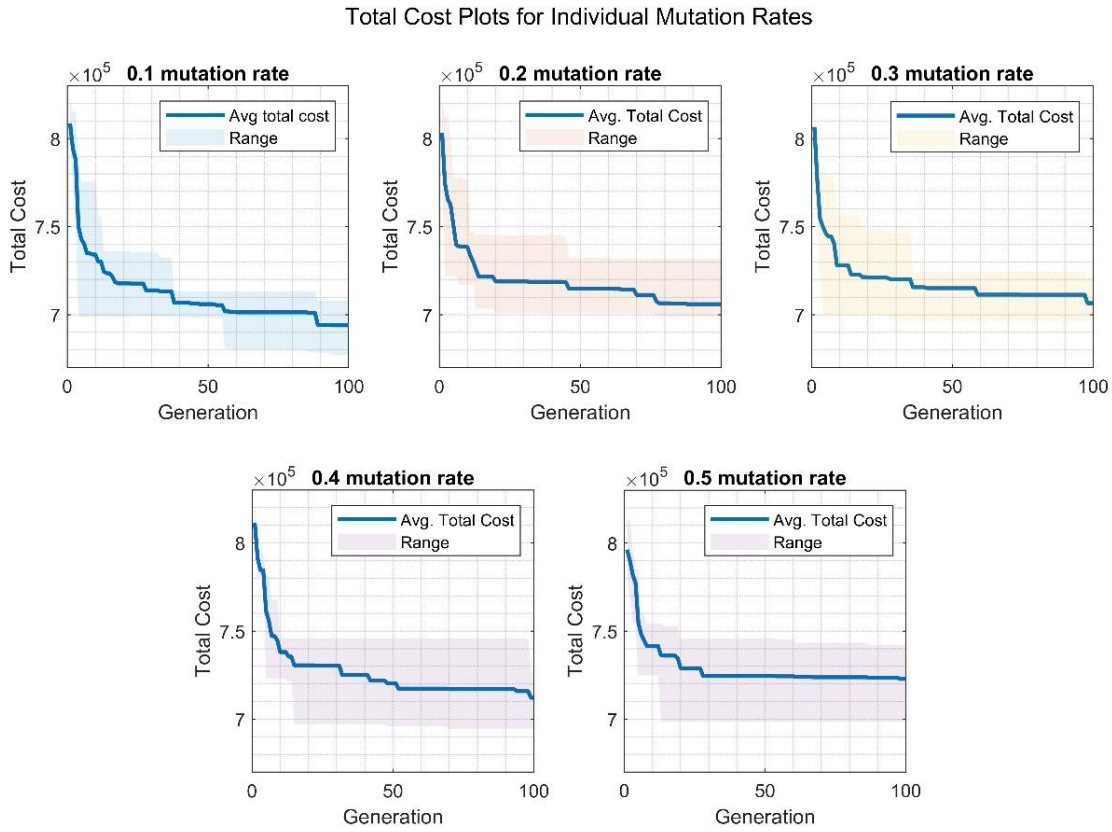


Fig. 5-15. Total cost plots for each mutation rate value tested. Shaded regions indicate min/max values obtained from the 5 simulation runs.

5.3.2 Tournament Size Testing

From the 2D testing of the cutting and packing algorithm, it was noted that using a large elite population resulted in diversity loss in the elite population, with very similar results frequently being retained across generations. To help mitigate this issue in the 3D algorithm, a smaller elite population size of 1 was used to ensure that the best solution found is preserved in each generation without the risk of the elite population becoming overpopulated with similar solutions.

With the elite population size fixed, the effect of varying the selection pressure (by changing the tournament size) on the algorithm's performance was investigated. Specifically, this investigation aimed to examine whether using a high selection pressure in combination with a minimal elite population size would lead to better solutions, or whether it would lead to premature convergence due to reduced diversity.

For testing, 3 different selection pressures were tested, 0.1 (low), 0.25 (medium) and 0.5 (high), which correspond to tournament sizes of 10%, 25% and 50% of the population size. Given that a population size of 20 is used for this section of the study, this corresponds to tournament sizes of 2, 5 and 10 respectively. As before, the algorithm was run 5 times for each selection pressure value, with the results averaged over the 5 runs. Additionally, a mutation rate of 0.1 was used.

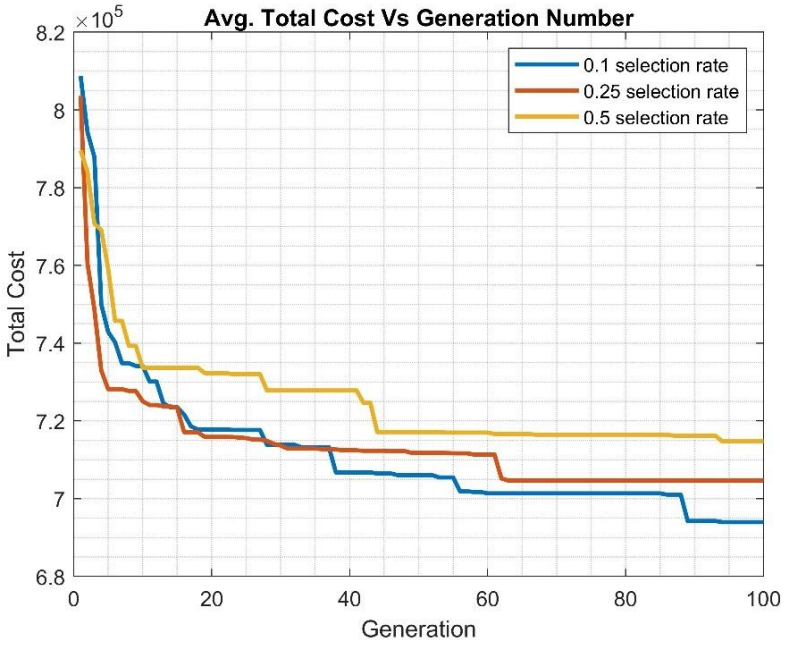


Fig. 5-16. Total cost (averaged over 5 runs) for each selection rate value tested.

Fig. 5-16 shows the total cost of the best solution in each generation (averaged over 5 runs) for each selection rate tested. From this figure, it is evident that the algorithms performance degrades as the selection pressure is increased, with the higher selection rates showing premature convergence to worse values. Looking at the individual plots for total cost in Fig. 5-17, it is also evident that the consistency of the algorithm degrades as well, with the ranges in total cost values across the 5 runs increasing with the selection pressure.

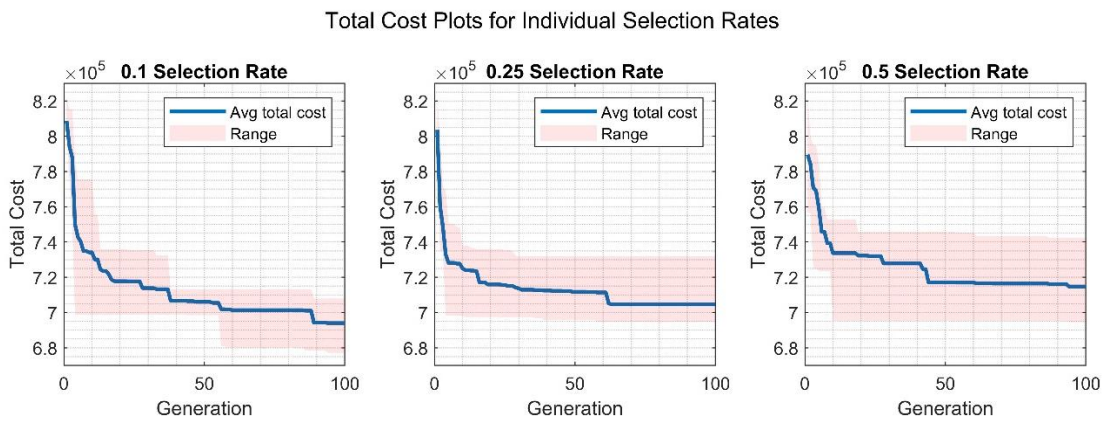


Fig. 5-17. Total cost plots for each selection rate value tested. Shaded regions indicate min/max values obtained from the 5 simulation runs.

It is well known in GA literature that increasing selection pressure will likely lead to worse performance, particularly through a loss of diversity in the population [204], [205]. This is due to the fact that a larger tournament size means that fitter individuals in the population will more consistently be selected for crossover and mutation, increasing the greediness of the search [145].

Another possible factor, closely related to this, which may also contribute to a degradation of the search quality is the problem of over exploitation of good solutions early in the search process. The problem with consistently selecting the best individuals for crossover and mutation is that it will highly favour these solutions early in the search. Unless the algorithm starts with strong solutions, this can lead to premature convergence to poorer local optima before the algorithm has a chance to explore other, potentially more promising, areas of the search space [149].

Fig. 5-18 shows the total cost of the best solution in each generation along with the average total cost of the entire population in each generation (averaged over the 5 runs), for each selection rate tested.

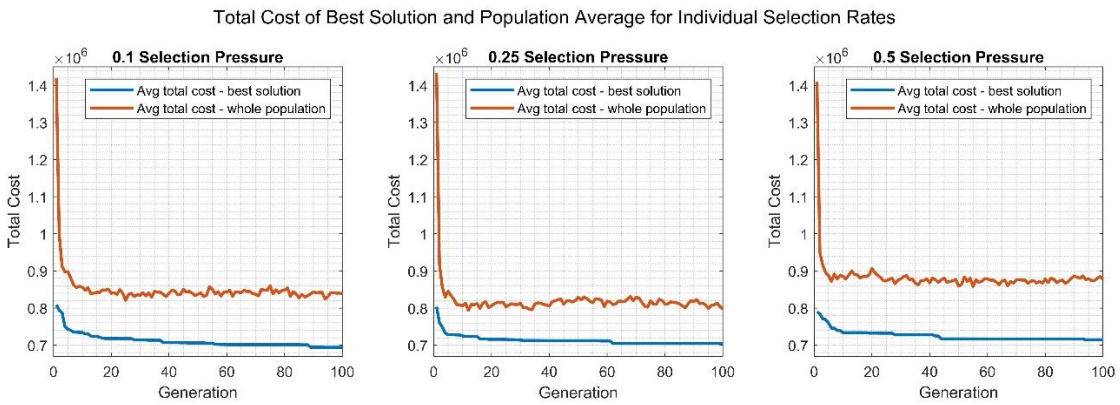


Fig. 5-18. Total cost plots for the best solution and the population average in each generation, for each selection rate value tested.

From this figure, it can be seen that, despite there being some variation in the average total cost of the population across the generations, the average cost remains relatively steady across the 100 generations. That is, the average population cost does not decrease throughout the search, indicating that the population has maintained good diversity throughout all the runs. In contrast, if the entire population was prematurely converging due to high selection pressure, it would be expected that the average cost of the population would steadily decline over time as the individuals become increasingly similar (i.e. as diversity is lost).

As such, it can be concluded that the observed drop in performance with increasing selection pressure is not due to a loss in diversity, but rather more likely due to over exploitation of poorer solutions early in the search. I.e. the high selection pressure makes the search heavily biased toward a small number of relatively good solutions early on, which prevents the algorithm from effectively exploring more promising areas of the search space as the search progresses.

As a final point, it is worth noting that Fig. 5-18 exhibits a markedly different y-axis (total cost) scale compared to the previous figures in this section, along with a sharp reduction in the population-average total cost within the first 2-3 generations. This behaviour is due to the population initialisation strategy used for the 3D algorithm (see Section 5.2.3 above). As discussed earlier, four population members are initialised using the ‘small’ cutting limit, resulting in four cutting patterns with many cuts (and correspondingly high cutting costs). In the first generation, these high-cost individuals inflate the population-average total cost. However, due to their poor fitness, these solutions are quickly eliminated by the selection process within the first few generations, resulting in the sharp drop and stabilisation of the population-average total cost.

5.3.3 Population Size Testing

For testing the population size, the algorithm was tested with population sizes of 20 and 100, with the algorithm being run 5 times for each population size. The motivation for this test stems from the high computational cost associated with solving the 3D cutting and packing problem. For example, the average run time for a single run during the mutation rate testing (which was done with a population size of 20) was approximately 391 minutes (around 6.5 hours). Given this level of computational expense, it was not feasible to test a wider range of population sizes due to time constraints. Instead, the focus of this study was on comparing a small and large population size to determine whether using a more conventional large population size would yield better results, and whether the increase in quality of solution would justify the additional computational overhead. For the results presented in Fig. 5-19, a mutation rate of 0.1 and a tournament selection pressure of 0.1 was used.

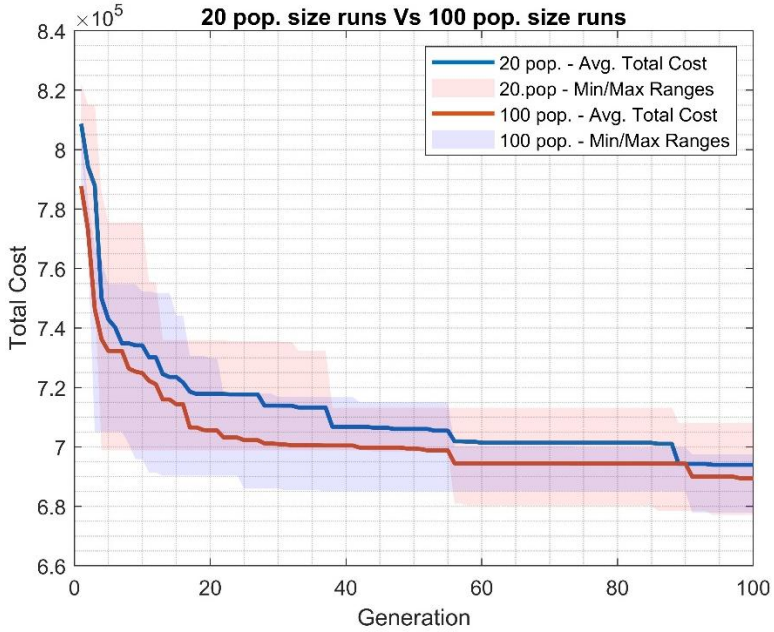


Fig. 5-19. Total cost (averaged over 5 runs) for the small and large population sizes.

From Fig.5-19, the algorithm's performance is visibly better on average for the large population case when compared to the small population case. For the 100-population case, the algorithm converges faster earlier in the search when compared to the 20-population case. It can also be seen that, on average, the 100-population runs converge more consistently to better solutions, with the final average cost being lower, and the range

of final cost values for the 5 100-population runs being narrower. This is unsurprising since having a large population promotes better exploration of the search space, helping to reduce the likelihood of premature convergence to local optima.

However, the key point to note with this comparison is that the improvement in average total cost between the two cases, while evident, is very minor. Looking at the average final total cost values for both cases (693,985 for the 20-population case and 689,478 for the 100-population case), the final average total cost for the 100-population runs is only 0.65% lower than for the 20-population runs. This suggests that even a small population size can still be effective for this problem when combined with well-tuned genetic operators.

This observation has significant practical implications when considering the computational cost. For example, the average run time for the 100-population runs was approximately 2,111 minutes (around 35.2 hours) which is an almost six-fold increase compared to the average runtime of 6.5 hours for the 20-population size runs. Given this significant increase in run time only resulted in minor improvements in the average total cost, it is evident that the increased computational expense is not worth this trade-off.

5.3.4 Hyperparameters for the Cutting and Packing GA

Based on the studies presented in this section, the parameters outlined in Table 5-1 are used for the remainder of this study (unless explicitly stated otherwise).

As a point to note, for the results presented in this section, and for the remainder of this report, zero-width cuts were used, and the cut parts were allowed to rotate 90 degrees about each axis during packing (giving 10 possible orientations; the starting orientation + 90, 180 and 270 degrees about each axis).

TABLE 5-1. Hyperparameters used for the cutting and packing GA.

Hyperparameter	Value
Population Size	20
Number of Generations	100
Number of Elites	1
Selection Method	Tournament Selection
Selection Pressure	0.1
Crossover Method	Random Single-Point Crossover
Mutation Method	Point Mutation
Mutation Rate	0.1
Thickness of Cuts	0
Packing Rotation Angle	90°

5.4 Simulation Setup

Given that information regarding specific nuclear decommissioning protocols and costing models is often proprietary and not accessible for this study, the focus is instead placed on demonstrating the flexibility (and examining the efficacy) of the proposed optimisation approach. Instead of a direct real-world comparison, the algorithm is applied to three hypothetical decommissioning scenarios, each involving different structures (commonly found in decommissioning), cut with different cutting tools and packed into UK industry-standard containers.

This study is designed to serve several purposes:

1. To illustrate how the algorithm can be applied to different structures (of varying size and complexity), using any type of container and cutting tool, to meet varying cutting and packing requirements.
2. To examine the algorithms efficacy when faced with different structures of varying size and geometric complexity.
3. To demonstrate how the algorithm can outperform a more conventional ‘minimal cutting followed by packing’ approach (where structures are partitioned using the 4 minimal cutting approaches listed in Section 5.2.3).

In addition to this, the efficacy of the proposed algorithm is further examined by:

1. Benchmarking against a manual approach (where a human user designs the cutting patterns).

2. Investigating the algorithms performance under different cutting and packing cost weight combinations.

5.4.1 Three Decommissioning Cases

For the three decommissioning scenarios presented in this paper, the three structure/container combinations depicted in Fig. 5-20 are selected: a glovebox (Fig. 5-20 a) packed into 500 L drums, and a small submarine RPV (Fig. 5-20 b) and heat exchanger pipe bundle (Fig. 5-20 c) both packed into 3 m^3 boxes. The mesh model for the heat exchanger pipe bundle was obtained from [100], the RPV model from [99] and the model of the glovebox was adapted from [206]. For the glovebox, UK industry standard 500 L stainless steel waste drums [102] are used to pack the cut parts. These drums are commonly used in nuclear decommissioning due to their mass-producibility, suitability for intermediate-level waste, and ability to be grouted for additional radiation shielding (as is common practice in intermediate-level waste disposal) [207]. For both the RPV and heat exchanger pipe bundle, 3 m^3 stainless steel boxes were used [102], which are also industry standard and often widely used for packing larger items of intermediate-level waste (e.g. large cut structural components for which 500 L drums would be unsuitable). These boxes offer the same advantages, being robust, mass-producible, and capable of being grouted for long-term disposal, again in line with standard disposal practice [207].

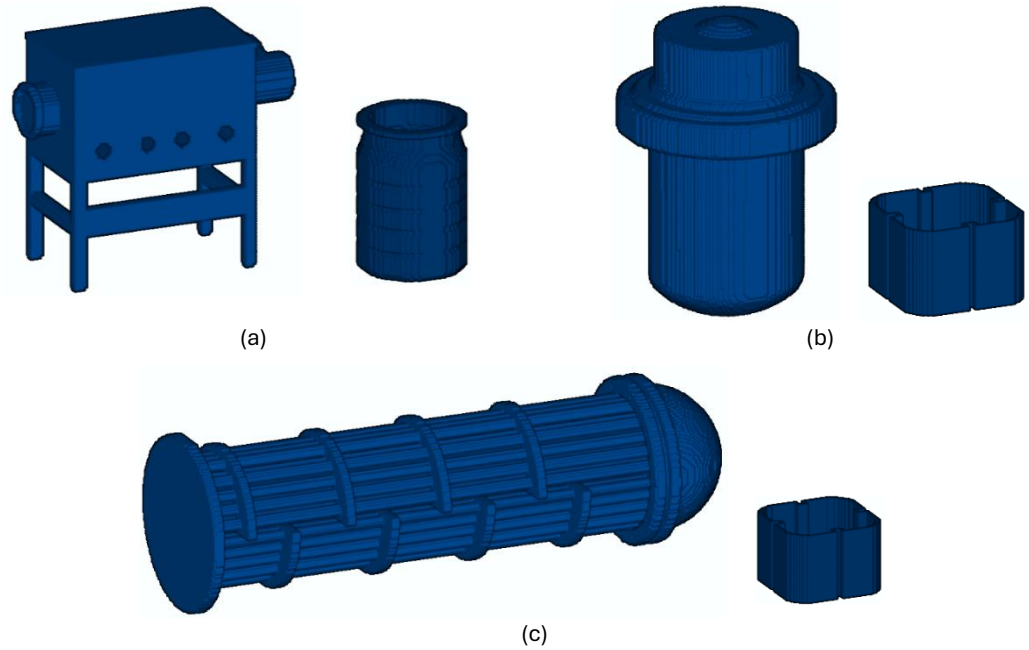


Fig. 5-20. Voxelised models of the three structures (and their respective containers) used in the three decommissioning scenarios. (a) – Glovebox and 500 L drum, (b) – Submarine RPV and 3 m^3 box, (c) – Heat exchanger pipe bundle and 3 m^3 box. Note, containers are shown to scale.

The voxel dimensions (given as l, w, h) for the three structures is: $(112,\ 48,\ 92)$ for the glovebox, $(90, 90, 128)$ for the RPV, and $(269, 82, 81)$ for the heat exchanger pipe bundle.

To determine the voxelised dimensions of the containers, the real-world dimensions of the structures needed to be established, which was then used to calculate the appropriate container scales. For the RPV, the height is provided as 4.2 metres in [99]. The other two models however, the glovebox and heat exchanger pipe bundle, did not include real-world dimensions. For these structures, the dimensions of similar real-world models was referenced: a glovebox structure with a height of 1.82 metres, as listed in [208], and a heat exchanger pipe structure with a length of 10.4 metres, as provided in [209].

With these values, and the dimensions of the voxelised structure models, the voxel size per centimetre for each structure was calculated (by dividing the real-world dimension value by its voxel counterpart), with the values for each structure listed in Table 5-2. Using the container height values given in [102], the container models are then scaled accordingly.

This gives voxel dimensions (l, w, h) of: $(43, 43, 63)$ for the 500 L drum container, $(57, 57, 39)$ for the 3 m^3 box used to pack the RPV, and $(47, 47, 32)$ for the 3 m^3 box used to pack the heat exchanger. For the part size limits (x, y, z) these were set to $(25, 25, 59)$ for the 500 L drum, $(46, 46, 36)$ for the 3 m^3 RPV container, and $(37, 37, 28)$ for the 3 m^3 heat exchanger container. A point to note; since the internal volume of the selected containers is not a perfect cuboid, the maximum size limits are set to match the largest cuboid that can fit within each container's internal packing space.

Regarding the choice of cutting tools, as with the structure dimensions, real-world examples where similar structures have been decommissioned were referenced. For the glovebox and heat exchanger, an electric hand saw and plasma cutter was selected respectively, as both tools have been successfully used to dismantle similar structures in [12] and [13]. For the Reactor Pressure Vessel (RPV), abrasive water jet cutting was selected, based on its common application in reactor pressure vessel segmentation [17], [210], [211]. [4] provides information on each of the selected cutting tools, listing operational costs as ‘low’ for electric hand saw cutting, ‘medium’ for plasma torch cutting, and ‘high’ for abrasive water jet cutting. Based on this, cost-per-cm values of 0.01, 0.05, and 0.1 were assigned to the electric hand saw, plasma cutter, and abrasive water jet cutting, respectively. With these cost values and the voxel size for each structure, the cutting cost weight, w_1, for each structure (shown in Table 5-2) is calculated by multiplying the two

values together. For the cost of the containers (w_2), values of 100,000 and 500,000 were assigned for the drum container and 3 m^3 box container respectively.

TABLE 5-2. Parameters for the three decommissioning scenarios.

Structure	Cutting Tool	Cutting Cost/cm	Voxel Size (cm)	W1	Container Type	W2
Glovebox	Electric Hand Saw	0.01 (low)	1.98	0.0198	500 L Drum	100,000
RPV	Abrasive Water Jet	0.1 (high)	3.23	0.323	3 m^3 Box	500,000
Heat Exchanger	Plasma Cutter	0.05 (medium)	3.84	0.192	3 m^3 Box	500,000

5.4.2 Benchmarking Against Manual Approach

For clarity, the term ‘manual’ in this context refers to a non-automated, software-based planning process carried out entirely within the decommissioning software NuPlant; no physical cutting experiments or real-world trials were undertaken as part of this study.

In the manual trials, a human user was tasked with defining 30 cutting patterns for each structure using an interactive interface within NuPlant. The user manually added and adjusted cuts (with cuts being limited to orthogonal cuts to ensure fair comparison to the GA), and once a pattern was defined, the resulting cut parts were processed through the packing algorithm. Afterward, the user reviewed the optimised cutting and packing solution, noting the individual cutting and packing costs, which helped inform the planning for the subsequent cutting pattern. Additionally, as with the GA approach, if the distance between two manually defined cuts along any axis exceeded the specified limit for that axis, the repair mechanism would automatically insert cuts between them to ensure the parts remained small enough to fit into the containers.

The motivation behind this study stems from the fact that, currently, users have only two options for planning a decommissioning strategy digitally: using a separate optimisation approach, like minimal cutting followed by packing optimisation on the cut parts, or manually defining cutting patterns and then applying packing optimisation. In addition to comparing the algorithm to the four minimal cutting approaches (discussed in Section 5.2.3), this test serves as a counterpart, allowing a comparison between the algorithm’s performance against the two existing approaches.

Additionally, this test also served as an insight into understanding how the human user approached the task. Given the intelligence and flexibility of a human user compared to a

stochastic algorithm like a GA, this analysis aimed to identify strategies adopted by the human user which might offer insights for future improvements to the algorithm.

5.4.3 Cost Weight Sensitivity Analysis

For the cost-weight testing the same simple RPV structure and cuboid container combination which was used for the hyperparameter testing (shown in Fig. 5-11) was used. The same structure and container was used for all the cost weight testing to give a consistent and controlled test scenario, enabling the effects of changing the cost weights to be examined in isolation without introducing variability from the structure of container geometry.

To test the algorithm's response to different cost weight combinations, a grid search is performed across a range of cost weight values. Specifically, combinations of cutting and packing cost weights were tested using the following sets of 5 values: cutting cost weights = [0.01; 0.1; 1; 10; 100], packing cost weights = [1,000; 10,000; 100,000; 1,000,000; 10,000,000].

The selection of these values was based on preliminary experiments which were used to establish a baseline where the cutting and packing terms contributed roughly equally to the total cost. For this structure and container combination, a cutting cost weight of 1 and packing cost weight of 100,000 was found to give total cost values where both cost terms were of similar orders of magnitude. With the baseline established, the effect of systematically increasing or decreasing each weight independently by orders of magnitude of 10 was explored. This allowed for an examination of how the algorithm performs when faced with cost trade-offs where either the cutting or packing is strongly prioritised (i.e. where one of the terms dominates the total cost).

The motivation for conducting this cost weight testing was twofold. First, it enabled an investigation into how the relative importance of cutting versus packing costs affects the algorithms behaviour and the quality of the solutions produced. Secondly, it provided an opportunity to examine the algorithms robustness under extreme cost weight scenarios (e.g. disproportionately high packing cost weights) to determine whether the optimisation methodology would still perform reliably or if the performance would degrade under such conditions.

5.5 Results

The following section presents the results and discussions for the test scenarios outlined in Section 5.4 above.

5.5.1 Three Decommissioning Cases

For the three decommissioning cases, the algorithm was run 5 times on each structure, with the results averaged over the 5 runs. The following section presents the results from these tests, and is structured in 4 parts:

1. **Total Cost Behaviour** – This section provides an analysis of the total cost curve plots produced by the GA, with particular focus on the consistency, convergence speed and run time of the algorithm for each structure tested.
2. **Individual Cost Behaviour** – This section examines in more detail how the individual cost elements of the cost function (cutting cost and packing cost) independently evolve over the course of optimisation, highlighting trade-offs between the costs and examining the convergence behaviour of the individual cost terms.
3. **Influence of Initialisation Strategy** – This section briefly examines the effect of the initialisation strategy (outlined in Section 5.2.3) on the best initial solutions found by the algorithm.
4. **Best GA Results** – This section presents the best solutions (lowest total cost) from the 5 GA runs, for each structure, with key metrics (e.g. number of cuts, containers and percentage utilisation of each container) highlighted. This section also discusses how structure complexity and the choice of cutting method influences packing performance for each structure.

Total Cost Behaviour

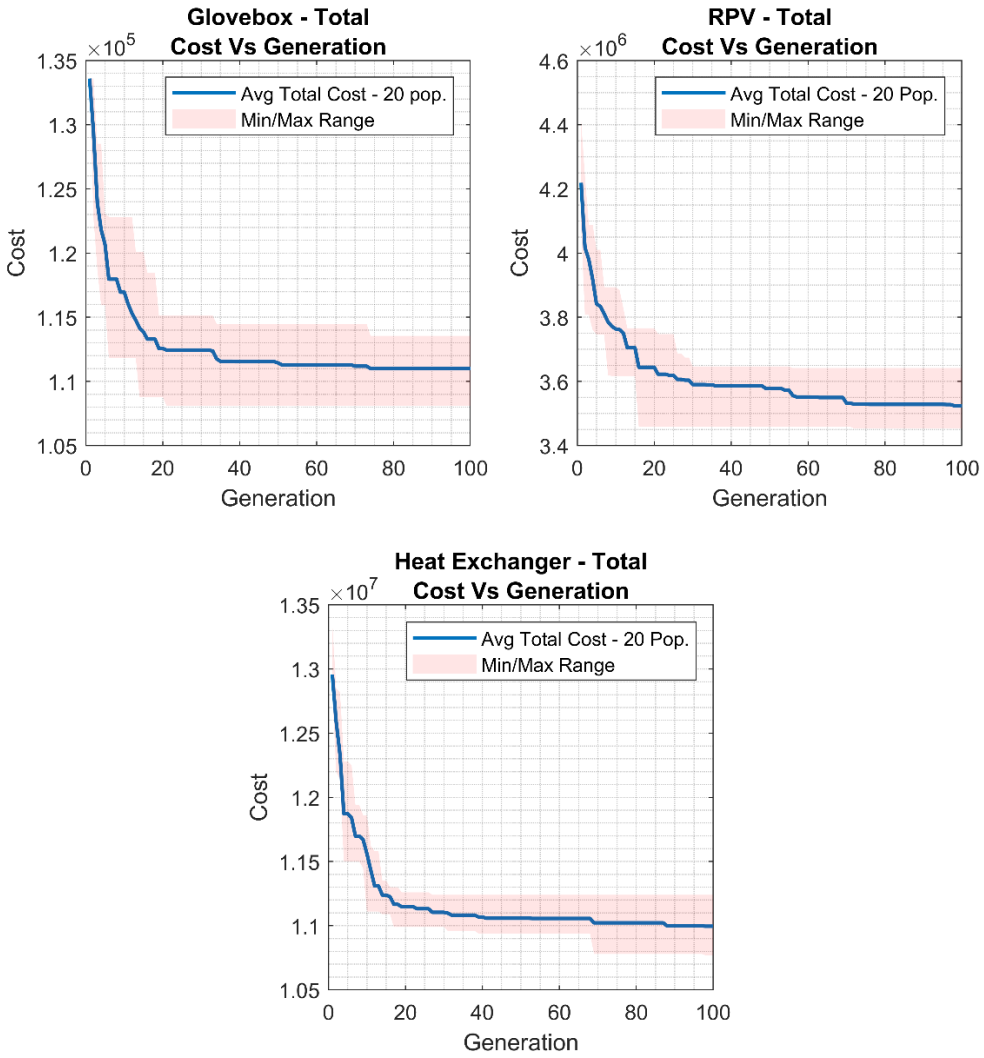


Fig. 5-21. Total cost plots for each structure tested. Shaded regions indicate min/max values obtained from the 5 simulation runs.

Fig. 5-21 above shows the total cost for the best solution in each generation, averaged over the 5 runs with the shaded regions showing the range (highest/lowest values across the 5 runs).

Over the 100 generations the GA was run for, an average cost reduction of 16.9% was achieved for the glovebox structure, 16.5% for the RPV, and 15.1% for the heat exchanger. Given that each of the 5 runs was initialised with four minimal-cutting approaches (where the part size limit is 1* x, y, z), the reduction in cost shows that the simultaneous cost optimisation approach proposed here can outperform a 'minimal cutting followed by packing' strategy. The problem with minimal cutting is that, whilst it helps keep cutting costs low, the focus on only reducing cutting cost leads to large cut parts which do not pack well,

leading to high packing cost and high total cost overall. This helps highlight the importance of considering both cutting and packing when planning optimised decommissioning strategies and suggests that using a cutting optimisation algorithm alone to minimise cutting (as in [18], [19]) is inadvisable as it will likely produce large, poorly packable parts.

To gauge the consistency of the algorithm across the 5 runs for each structure, the relative range (spread) in total cost at generation 100 is calculated for each structure as:

$$Spread\ (\%) = \frac{\max(total\ cost) - \min(total\ cost)}{mean(total\ cost)} * 100 \tag{9}$$

This metric reports the width of the band which encloses the final five cost values, normalised over the average. For example, for the glovebox structure, a value of 4.9% was obtained, indicating that that the final highest and lowest total cost solutions across the 5 runs differ by only 4.9% of the mean final cost (i.e. every run finishes within approximately $\pm$2.5% of the mean). The spread for the three structures tested is reported as: 4.9% for the glovebox, 5.4% for the RPV and 4.3% for the heat exchanger, meaning that overall, the best and worst final costs differ by approximately $\pm$2-3% of their mean. Given the small values obtained for the three structures, and the consistency across them, this indicates that the algorithm has high repeatability (i.e. can obtain consistently good solutions) across a range of structure complexities.

To gauge the convergence speed of the algorithm, the number of generations required for the average total cost to converge to within 5% and 1% of the final average total cost is calculated for each structure. The number of generations required to converge within 5% of the final total cost is reported as: 11 generations for the glovebox, 16 generations for the RPV and 11 generations for the heat exchanger. Likewise, the number of generations required to converge within 1% is given as: 34 generations for the glovebox, 56 generations for the RPV and 27 generations for the heat exchanger. This shows that the algorithm is able to capture $\geq$ 95% of attainable savings in cost within the first 10-20 generations, and $\geq$ 99% within 20-60 generations, with any cost gains after this being marginal.

The implication for this is that in future, if reduction in computation time is needed, the algorithm could be augmented with an early stopping criterion (e.g. stop when the current generation best cost is within 1% of the best-found cost after 10 consecutive generations) without losing much in terms of cost savings. Relating to the 3 cases discussed above, if such a criterion were to be implemented, it could cut the run time by a factor of 2 – 4 times whilst still obtaining results almost indistinguishable from the results obtained after 100 generations.

The average run times for the three structures is reported as 1.98 hours for the glovebox, 14.40 hours for the RPV and 60.98 hours for the heat exchanger. The primary reason or the increase in computation times across the 3 structures can be attributed to the packing side of the algorithm. With larger structures, more containers are required to pack the cut parts. Recall that with the packing method chosen for this study (PBP), whenever an allocated set of parts fails to be fully packed into a container, the algorithm executes a 'mending' step where another subset of parts is allocated to the partially packed container. Given that the average number of containers needed to pack the heat exchanger was 16.6, compared to only 6 containers for the RPV and just 1 container for the glovebox, the significant increase in run time for the heat exchanger comes primarily from the larger number of containers required. More containers means the packing algorithm will need to perform the mending step more times, increasing the computation time needed to successfully pack all the cut parts.

Table 5-3 below summarises the key statistical results discussed in the total cost analysis presented above.

TABLE 5-3. Summary of key metrics from total cost analysis.

Structure	Run Time (hrs)	Avg. Total Cost Reduction (%)	Spread (%)	Gen. of < 5%	Gen. of < 1%	Avg. Num. Containers
Glovebox	1.98	16.9	4.9	11	34	1
RPV	14.40	16.5	5.4	16	56	6
Heat Exchanger	60.98	15.1	4.3	11	27	16.6

As a final point to note, regarding the larger number of generations required to reach the 5% and 1% convergence threshold for the RPV structure, this is likely due to the geometry of the structure and resultant cut parts. For the glovebox and heat exchanger, the more 'irregular' geometry of the structures allows the cutting algorithm to produce a wide range of cut parts in terms of their sizes. This makes the quality of the packing solutions less sensitive to the cutting, since the large parts can be packed first, with gaps being filled by smaller parts. In contrast, for the RPV, the cut parts tend to be larger and more uniform in size, with fewer small parts available to fill gaps in the packing structure. As such, the quality of the packing solutions becomes more sensitive to the specific locations of cuts, with small adjustments to the cuts easily affecting how well the 'blocky' cut parts slot together in the packing space. This increased sensitivity likely contributes to the longer time taken for the GA to find good

solutions for the RPV structure. An illustration of the differences in part uniformity is given in Fig. 5-22, which shows a packed container for the RPV (left) and the glovebox (right).

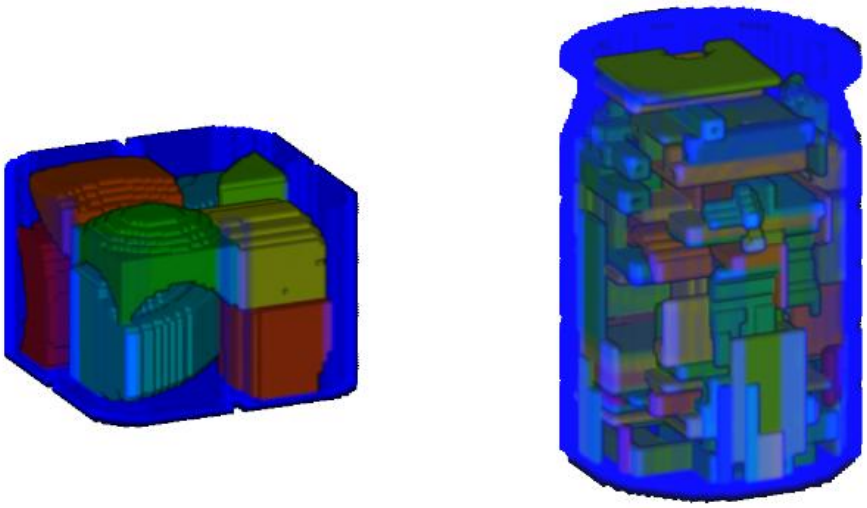


Fig. 5-22. Illustration of difference in part uniformity for cut parts from the RPV (left) and from the glovebox (right).

Individual Cost Behaviour

Fig. 5-23 (on the subsequent page) shows the individual cutting and packing costs for the best solution in each generation, averaged over the 5 runs, with the shaded regions showing the range between the highest and lowest values obtained.

From these plots, there are several observations which can be made:

Firstly, trade-offs between the cutting and packing costs can be seen as the generations progress. When a drop in packing cost occurs (via a reduction in the number of containers), this is typically accompanied by a rise in the cutting cost. This is particularly evident for the heat exchanger case, which shows an almost oscillatory change in the cutting and packing costs as the algorithm converges in the first 0-30 generations. In such cases, even though the cutting cost increases (due to an increase in the amount of cutting), this rise in cutting cost is outweighed by the reduction in packing cost, resulting in a net decrease in total cost overall. These observations show that the algorithm can balance the two conflicting cost objectives and find optimised trade-offs.

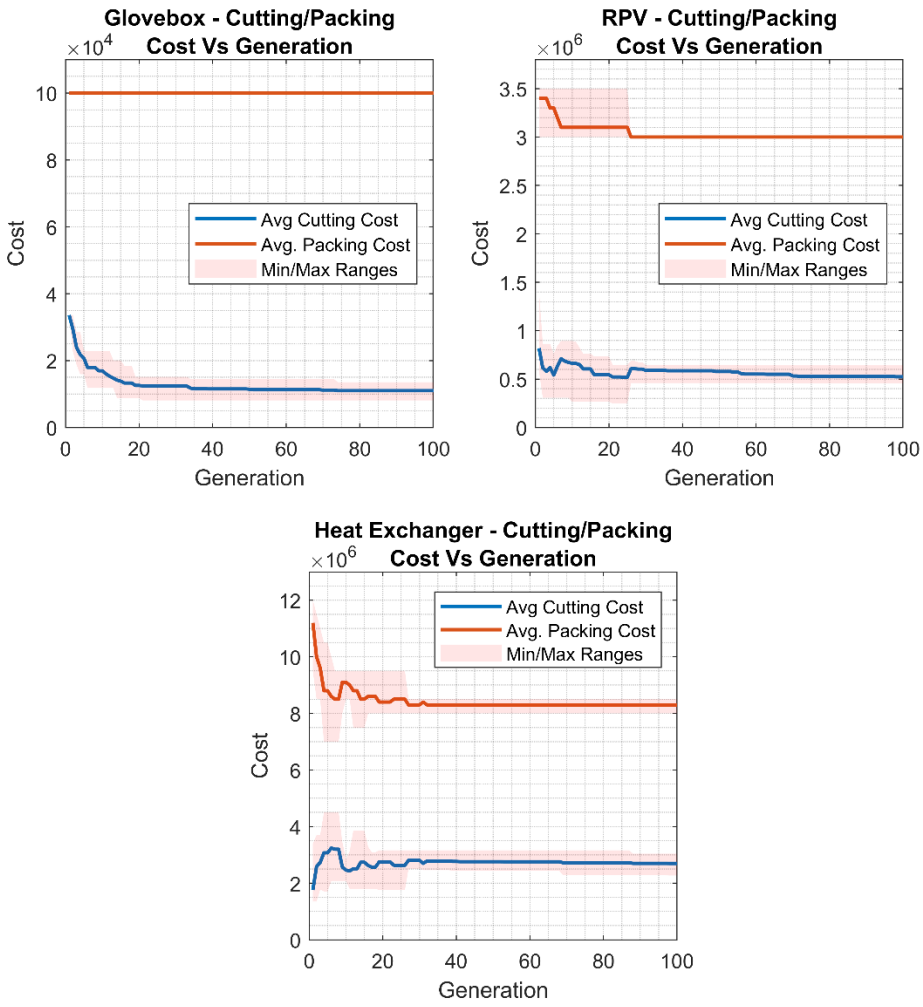


Fig. 5-23. Cutting and packing cost plots for each structure tested. Shaded regions indicate min/max values obtained from the 5 simulation runs.

Secondly, it can also be seen that the packing costs tend to converge early on, with the remainder of the cost reduction occurring from improvements in the cutting patterns. Specifically, for the heat exchanger and RPV cases, the packing costs converge within 0-30 generations. For the glovebox, all the runs managed to find the optimal solution (in terms of the number of containers) right from the start, giving a constant packing cost throughout. This observation reinforces the previous observation that the majority of large changes in cost tend to occur early on (within approximately 0-50 generations), with cost reductions after this being marginal.

It is also worth noting that for both the glovebox and RPV, the algorithm converged to the same number of containers in each run. In the heat exchanger case, three out of the five runs converged to 17 containers with the remaining two converging to 16 containers. Although this shows the algorithm's variability for more complex structures, this is still a small variation of only 1 container, showing that the algorithm can effectively estimate the number of containers required for a decommissioning task to a high degree of accuracy.

One possible theory for the early convergences observed in packing cost could be due to a loss in population diversity, preventing the algorithm from finding better solutions. However, it was noted in the selection rate testing that, even with the highest selection pressure, the population diversity (measured as the average total cost across the population in each generation) did not converge toward the best cost value in each generation over 100 generations. This observation suggests that diversity loss is not the likely cause for this early convergence behaviour.

The more likely explanation is that the algorithm has converged to the optimal (or near optimal) number of containers within the capabilities of the packing algorithm. It is worth emphasising that 'optimum number of containers' here refers to the best achievable solution given the limitations of the packing approach used. Although the packing algorithm selected (PBP) uses an allocation strategy which aims to minimise greedy behaviour, the packing process is still ultimately a greedy constructive process, since it: a) focuses on constructing a solution from scratch and then stops, without trying to optimise it further, and b.) restricts the packing order from largest volume to smallest. As such, using a more advanced algorithm (such as the hyper-heuristic algorithm proposed in Chapter 4, Section 4.2.3) could potentially allow the algorithm to find better solutions. The downside however, as discussed in Chapter 4, Section 4.2.4, is that the incorporation of a metaheuristic algorithm to optimise, for example, the packing order or choice of placement heuristics, would inevitably lead to much longer computation time, potentially making it impractical.

Another potential option could be to run a secondary post-packing improvement step to try and improve the solutions produced by the PBP algorithm. For example in the paper where the PBP algorithm was proposed ([176]), the authors also propose a secondary improvement step, referred to as 'Local Search 2' (LS2), which takes items from the least utilised containers and tries to re-pack them into other, more filled containers. They noted that, for all the datasets tested, LS2 managed to improve the packing score by 3.99% on average. The downside however is that it required much longer to run compared to the PBP algorithm alone. The average run time across all datasets for just PBP is given as 116 seconds, with the average run time for the LS2 procedure being 1158 seconds. This is an increase in run time of approximately 898% for only a 4% improvement on average. As such, using a procedure such as this, whilst potentially providing better solutions, would dramatically increase the runtime of the cutting and packing algorithm for only marginal gains in packing.

As such, future work should focus either on ways to intelligently improve the packing process without having to resort to metaheuristic optimisation techniques to trial multiple

packing solutions, or on a fast improvement step to improve packing further without adding significant computational overhead.

Influence of Initialisation Strategy

As outlined in Section 5.2.3, the initialisation strategy adopted for the GA was a combination of fixed cutting patterns (initialised with 'small', 'medium' and 'large' size limits), with the rest being randomly generated. The goal of this was to ensure that the initial population of the GA contained a balanced distribution of cut part sizes to try and reduce the likelihood of convergence to poor solutions due to poor initialisation.

When examining the best initial solutions for the five runs for each structure, several observations were noted. For the glovebox structure, each of the 5 best initial cutting and packing solutions were obtained from fixed initialisation cutting patterns. For the heat exchanger, 4 of the 5 best initial solutions came from fixed initialisation with one from random initialisation. However, for the RPV, 4 of the 5 best initial solutions came from random initialisation, with only 1 solution from fixed initialisation.

This suggests that, whilst the chosen initialisation strategy can provide good initial cutting patterns, random initialisation can still yield superior starting solutions. Furthermore, the solutions created using fixed initialisation take up a large portion of the initial population (12 of the 20 initial population members). For the structures where fixed initialisation provided the best initial solutions, this may not be an issue, however for the RPV case, if such a large portion of the population is initialised with poor solutions (i.e. worse than randomly generated ones), it could potentially limit the algorithm's ability to explore better ones.

As such, future work should focus on trying to reduce the number of fixed initial solutions to allow more randomness in the initial population, helping to improve the balance between exploration and exploitation. Additionally, work should focus on more intelligent initialisation strategies rather than using the simplistic minimal cutting approaches employed here, to ensure that they provide a stronger starting point for the algorithm.

Best GA Results

Fig. 5-24, 5-25 and 5-26 show the best total-cost optimised solutions obtained from the 5 runs for the glovebox, RPV and heat exchanger, respectively. The key metrics relating to

these solutions (number of cuts and cut parts, number of containers required, average percentage utilisation across the containers and run times) are also reported.

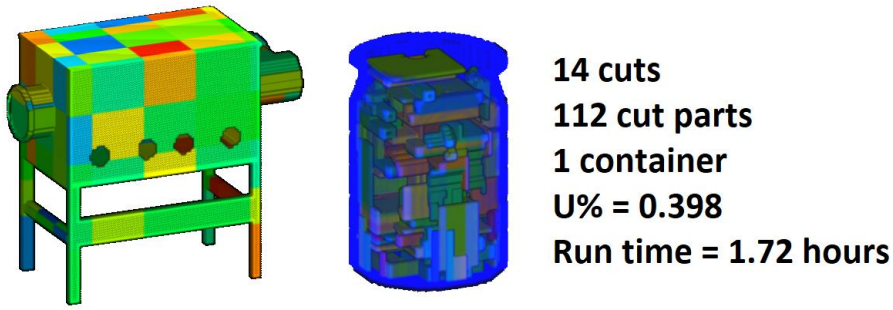


Fig. 5-24. Best solution found from the 5 glovebox runs.

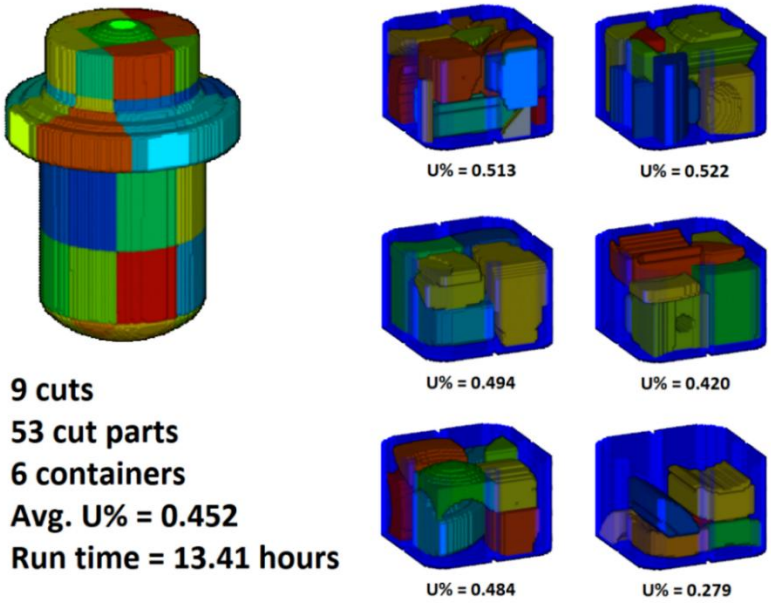


Fig. 5-25. Best solution found from the 5 RPV runs.

From these figures, additional observations can be made.

For the glovebox structure, the algorithm has successfully packed all the cut parts into a single container. Given that all parts fit into a single container, the utilisation score achieved is the highest possible for this structure and container combination. It can also be seen that the orthogonal cutting scheme is effective for this structure, producing mainly flat, regular shaped parts which stack efficiently in the container.

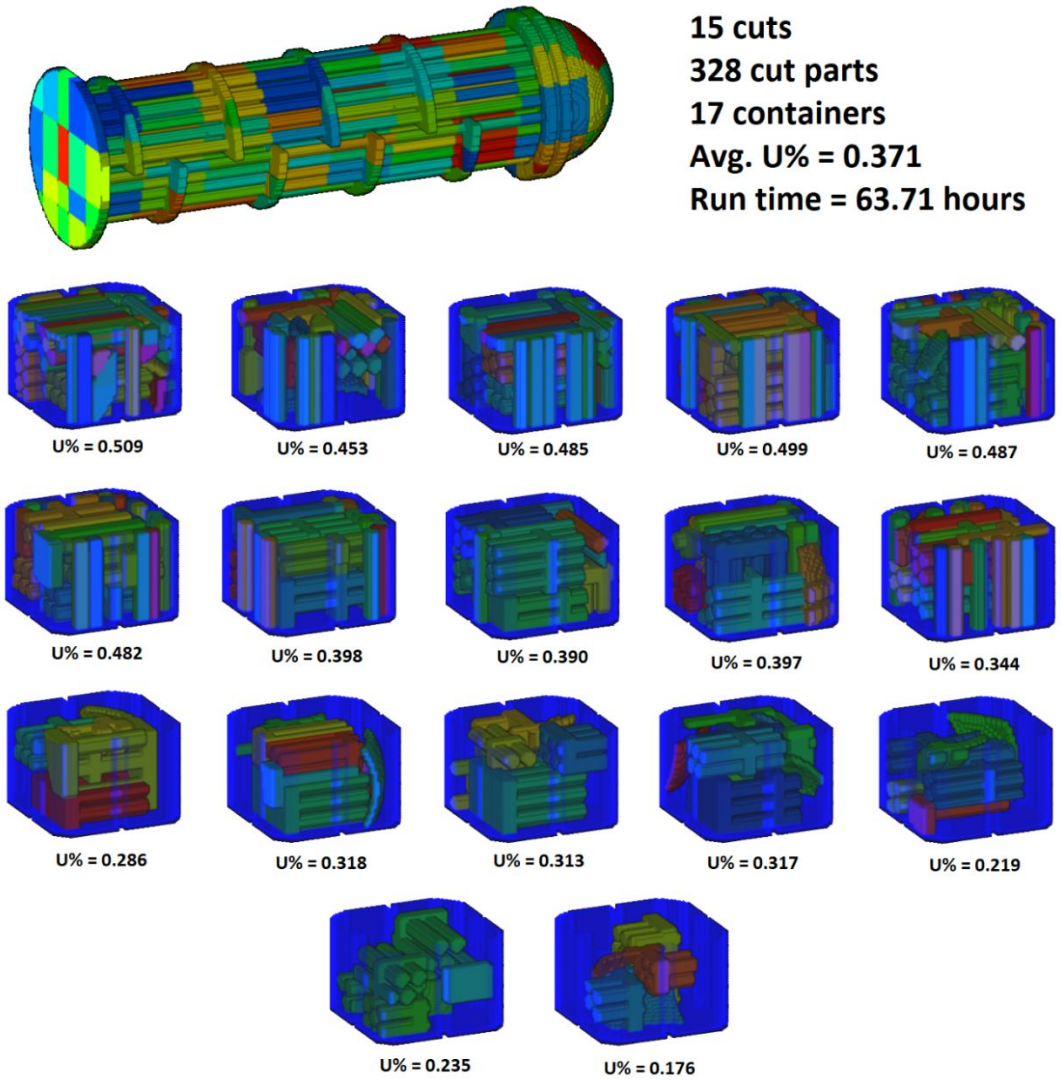


Fig. 5-26. Best solution found from the 5 heat-exchanger runs.

For the RPV structure, the orthogonal cutting scheme also shows good performance, producing many flat sided, evenly sized blocks which fit together well in the containers. This results in good consistency across the packed containers with a high average utilisation score. As an example, in [83], the authors present a multi-container packing algorithm for packing segments of a manually-partitioned RPV model. Their model is partitioned into 126 cut parts which are then packed into three large containers, with an average utilisation score of 0.297 (29.7%). In contrast, in the solution presented in Fig. 5-25, the average utilisation across the 6 containers was 0.452 (45.2%). Whilst this is not intended as a direct benchmark comparison (since they do not consider both cutting and packing optimisation), it highlights the difficulties with achieving high utilisation scores in the packing of irregular objects.

Finally, examining the solution for the heat exchanger structure revels additional problems with increasing problem complexity. Looking at the individual utilisation scores for the

containers, a noticeable drop in the scores can be seen as the packing progresses through subsequent containers (with the first and last containers having scores of 0.509 and 0.176 respectively).

Despite the allocation algorithm using an objective function which aims to minimise greedy behaviour (where small parts are grouped together early on leaving large parts till the end), the packing solution still shows a greedy trend. The problem is that, early in the packing process, the allocation algorithm has access to a broad range of parts in terms of their sizes, allowing for well-balanced allocations (large parts which are packed first, with gaps being filled by smaller parts, such as individual cut pipe segments). However, as the packing process advances to subsequent containers, the number of smaller cut parts quickly gets depleted, leaving mostly larger, more irregular shaped parts. As a result, despite the allocation algorithms attempt to minimise greedy behaviour, it cannot do so since there is no longer a good variety of parts left to pack. As stated previously, this problem could potentially be alleviated by running a secondary improvement step to try and re-pack the parts in the most poorly utilised containers, however doing so would incur additional computational overhead.

Another possible limitation could be with the cutting approach itself. Although the orthogonal cutting approach performed well for the glovebox and RPV, for the heat exchanger, it can be seen from Fig. 5-26 that it struggled to consistently separate the individual pipe segments from the partitioning baffles. For the cases where pipe segments were entirely separated from the baffles, the packing algorithm was able to effectively fill the gaps between larger parts with these smaller pipe segments. However, for cases where the pipe segments were not separated, it resulted in larger, more irregular parts which were more challenging to pack (this is more evident from the last few containers in Fig. 5-26).

In real world decommissioning, heat exchangers are typically dismantled by separating the pipe segments from the partitioning baffles (as shown in Fig. 5-27). If the algorithm could consistently replicate this real-world separation process, it would result in an increased number of smaller individual pipe segments. This could potentially mitigate the aforementioned problem observed with the allocation approach (where it runs out of small parts during the packing of later containers), helping to improve the overall packing efficiency.

Fig. 5-27. Real-world heat exchanger cutting example showing the pipe segments separated from the partitioning baffle. Image from [212].

5.5.2 Benchmarking Against Manual Approach

The following section presents the results from the 30 manual (non-automated, software-based) trials for each structure, and is structured in 2 parts:

1. **Comparison to GA** – This section examines the best results obtained from the manual trials for each structure and compares them to the results obtained from the GA, with particular focus on the differences in solution quality and run times.
2. **Analysis of User Strategy** – This section provides an analysis of how the human user approached the task, with observations drawn from the progression of solution quality over the 30 trials for each structure. Particular attention is given to the sensitivity to structure complexity and trends in user behaviour (in particular, the use of geometric features on the structures which the user used to strategically place certain cuts)

Comparison to GA

Fig. 5-28 shows the best total cost solutions obtained from the 30 manual trials for each structure, alongside the averaged results from the 5 GA runs (with population size of 20), with an additional GA run performed using a population size of 100. The large population size runs were conducted after the manual trials to test whether the use of a larger population size would enable to algorithm to outperform the human user.

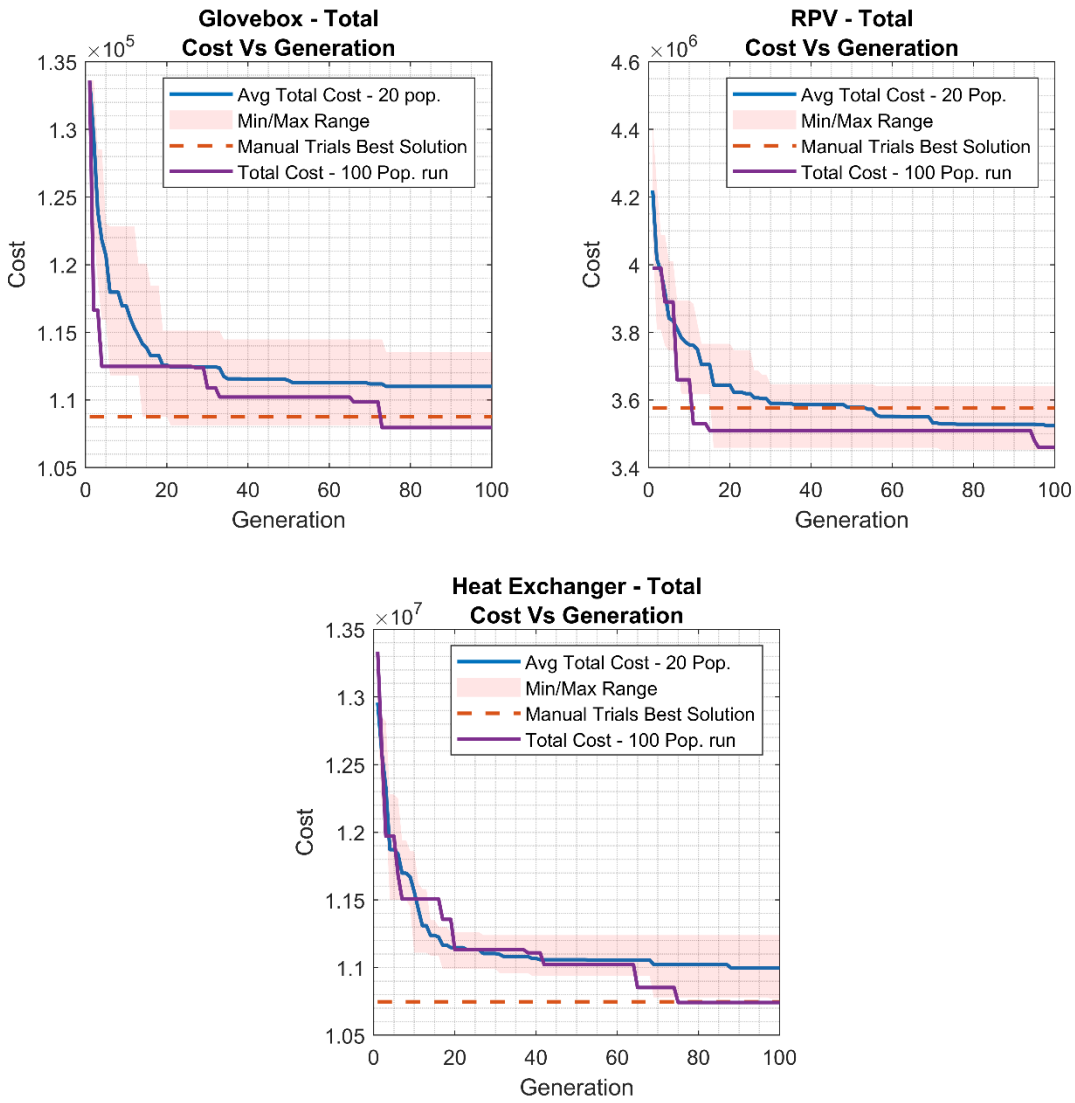


Fig. 5-28. Total cost plots for each structure tested showing total cost for: 5 20-population size GA runs (shaded regions showing range), best total cost from manual trials, total cost for 100-population size GA run.

Based on these results, several observations can be drawn:

- **Glovebox:** For the glovebox structure, the best total-cost result achieved by the human user outperformed the average total cost for the 5 20-population GA runs, with the total cost from the best manual trial being 2.03% lower than the final average total cost from the 5 GA runs. However, it can also be seen that the best manual result falls within the shaded range band for the 5 GA runs, indicating that the 20-population size GA can outperform the human user, although not consistently. For the 100-population size run, the final total cost was lower than for the best manual trial, however the improvement was only marginal, with a percentage decrease of 0.73% in total cost compared to the manual trial. This supports the idea that using a large population can achieve better results (compared to both the 20-population size GA and manual trials), but only by marginal amounts.

- **RPV:** For the RPV, the best total cost achieved by the human user was worse than the average total cost for the 5 20-population GA runs, with the total cost from the best manual trial being 1.50% higher than the final average cost from the 5 GA runs. However, as with the glovebox trials, the best manual result also falls within the shaded range band for the 5 GA runs, indicating that the GA can produce results worse than the human user. For the 100-population run, the final total cost value was also lower than the best manual trial, with a decrease of 3.38% in total cost compared to the best manual trial. It was noted previously that the convergence time for the RPV was larger compared to the glove box and heat exchanger, with the theory being that the higher uniformity in cut parts made the algorithm more sensitive to smaller changes in the cutting patterns. The difficulty for the human user to match the GA for the RPV can be attributed to this theory. I.e. given the sensitivity of the solutions to changes in the cutting pattern, similar to the GA, the human user struggled more with this structure.

- **Heat Exchanger:** The most significant difference between the manual trials and GA trials can be seen for the heat exchanger. For this structure, the best result from the manual trials is 2.26% lower than the final average cost from the 5 GA runs. Additionally, the best manual trial is just outside the shaded region for the 5 GA runs, showing that for this structure, the human user managed to outperform every GA run. For the 100-population run, the final total cost value obtained was almost identical to the best solution from the manual trials, with the final cost from the 100 population run being only 0.05% lower than the best manual trial. This shows that the GA struggles more for the complex structure with a small population but can achieve comparable solutions to the human user when the population size is increased.

Regarding the time taken to perform the 30 manual trials for each structure, the manual trials were not precisely timed, however the human user noted that it took approximately 1.5-2 working days (~12-16 hours) to complete the 30 trials per structure. In comparison, the average run times for the 5 20-population GA runs was 1.98 hours for the glovebox, 14.40 hours for the RPV and 60.98 hours for the heat exchanger. For the 100 population size runs, the run times increased substantially to 8.88 hours, 72.87 hours and 325.86 hours respectively.

Although the GA required considerably longer computation times (especially for the 100 population runs), it is worth noting that, unlike the manual trials, the GA does not require any form of human interaction throughout the optimisation process. From a practical perspective, while the GA may take longer to obtain solutions, it removes the need for a

human operator to manually define cutting patterns, helping to free up human resources for other tasks.

Table 5-4 below summarises the key results discussed in the comparison between the best solutions obtained from the manual trials and GA runs, as discussed above.

TABLE 5-4. Summary of key metrics from comparison between manual trials and GA.

Structure	Best Manual vs 20-pop GA Avg.	100- pop GA vs Best Manual	20-pop GA Avg. Run Time	100-pop GA Run Time	Manual Time
Glovebox	Manual 2.03% better	100-pop GA 0.73% better	1.98 hrs	8.88 hrs	~ 12-16 hrs
RPV	20-pop GA 1.50% better	100-pop GA 3.38% better	14.40 hrs	72.87 hrs	~ 12-16 hrs
Heat Exchanger	Manual 2.26% better	100-pop GA 0.05% better	60.98 hrs	325.86 hrs	~ 12-16 hrs

Overall, given the small percentage differences in total cost between the results obtained by the human user compared to the GA, it can be concluded that the algorithm can produce results comparable to a human (especially if the algorithm is run with a larger population). Whilst the computation times are generally larger for the GA, the closeness of the GA results to the manual results demonstrate that the algorithm is a practical alternative to time-consuming manual approaches, especially if longer computational times are acceptable.

Analysis of User Strategy

Fig. 5-29 (on the subsequent page) shows the progression of total cost across the 30 manual trials conducted by the user for each structure.

For the glovebox the user quickly identified a solution which was able to pack all the cut parts into a single container. However, once this optimal one-container solution was found, the user struggled to reduce the cutting cost further without degrading the packing solution. This can be seen in the fluctuations on the total cost plot for the glovebox. During these ‘jumps’ in total cost, the user was attempting to reduce the cutting cost by removing or altering existing cuts. The problem is that this often resulted in the cut parts no longer fitting into a single container (with parts spilling over into a second one), leading to sharp increases in total cost due to a large increase in packing cost. The user would then try to reintroduce cuts in an attempt to achieve a single container packing solution again, resulting in the distinct oscillations seen in the total cost plot. This observation shows that,

once the user identifies promising solutions (in terms of the number of containers), it becomes difficult to reduce the cutting cost further without inadvertently degrading the quality of the packing solution.

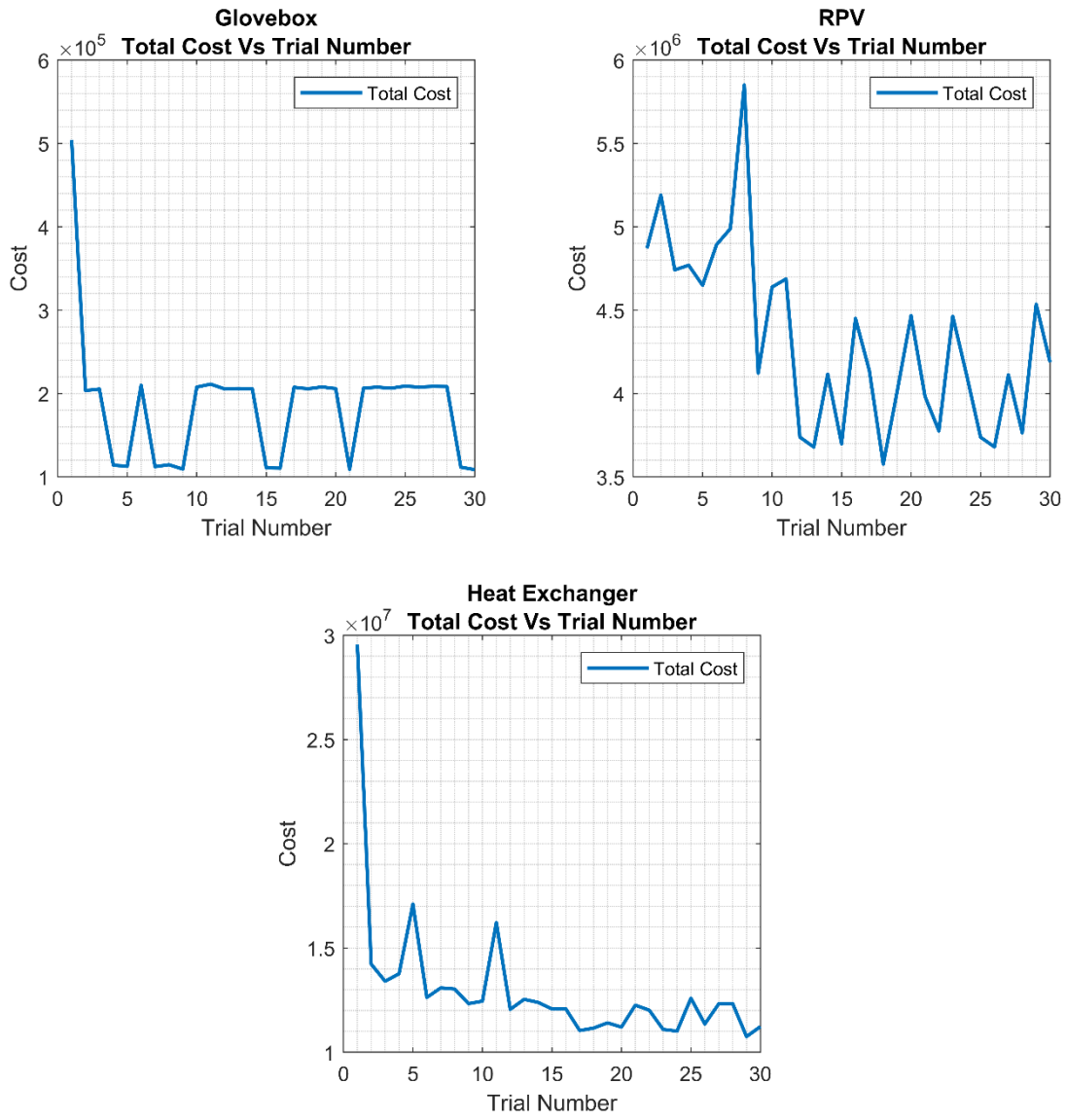


Fig. 5-29. Total cost of each manual trial, plotted against the 30 trials conducted by the user, for each structure.

For the RPV, the graph clearly shows greater fluctuation in the total cost across the 30 trials when compared to the other two structures. This reinforces the previous observation that this structure is more challenging due to the sensitivity to minor changes in the cutting pattern. The large block-like parts produced from the RPV are highly dependent on the exact locations of cuts, with even small changes to the cuts resulting in parts which no longer fit together well in the packing space. This larger sensitivity makes in more challenging for the

human user to refine the solutions over the 30 trials, since even small changes to the cuts can have a large impact on the quality of the overall solution.

For the heat exchanger, the total cost shows a more consistent downward trend across the 30 trials, showing that the user was able to incrementally improve the solutions. Given the larger number of containers required to pack all the cut parts (typically 16-17) coupled with the larger distribution in cut part sizes, the packing solution is less sensitive to minor changes in the cutting pattern (with adjustments to cuts having little or no effect to the number of containers). This allows the user to make incremental changes to improve the cutting cost with lower risk of dramatically affecting the quality of the packing solutions.

As a final point to note, an interesting observation made during the manual trials was the tendency of the user to place certain 'feature-based' cuts early one, which remained largely unchanged over the 30 trials. Referring to Fig. 5-30 as an example, for the glovebox, the user often placed cuts at the edges of the glovebox, to remove the top and sides, and to separate the top box section from the supporting legs. For the heat exchanger, the user often placed cuts at the edges of the partitioning baffles (to separate the pipe segments) and at the end of the structure to separate the pipes from the domed cap.

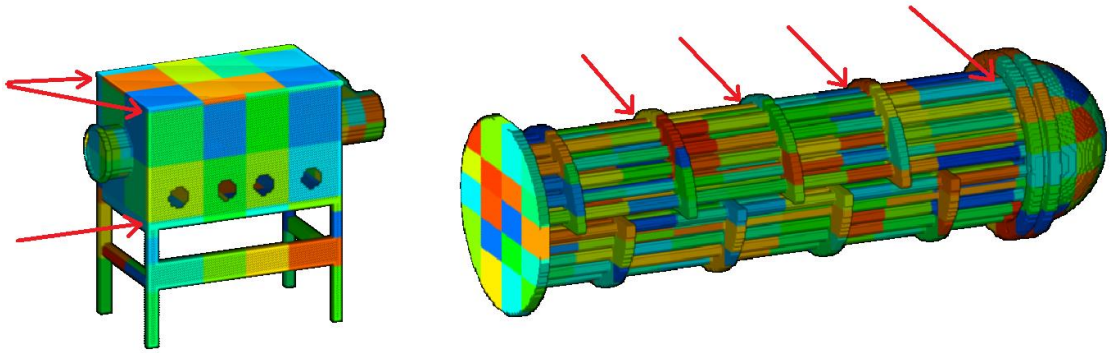


Fig. 5-30. Examples of 'feature-based' cuts applied to the glovebox and heat exchanger.

To a human user, the placement of cuts in these locations presents a logical choice based on the features of the structures. For the glovebox, separation of the top and sides allows these parts to be cut further into smaller flat segments which stack well in the packing space. For the heat exchanger, separation of the pipe segments from the partitioning baffles and end cap allows the smaller pipe segments to be efficiently placed into smaller gaps in the packing structure. This suggest that some cuts are more 'important' or 'natural' due to inherent features of the object.

The implication of this observation is that it could potentially be used to reduce the number of optimisation variables in the cutting and packing algorithm, helping to speed up the

search process. For example, future work could focus on developing a pre-processing feature-based partitioning scheme which attempts to separate the structure into primary parts based on cuts which mimic the human approach to feature-based cutting. Once placed, these cuts would not need to be optimised further, leaving the optimisation algorithm to focus more on 'uncertain' cuts (cuts where the effect on the solution quality is more uncertain).

5.5.3 Cost Weight Sensitivity Analysis

The following section presents the results and discussion for the cost weight sensitivity testing. For this experiment, the algorithm was run once for each cost weight combination shown in Table 5-5, for 100 generations, using the same RPV and container combination used for the hyperparameter tuning (Fig. 5-13).

Table 5-5 shows percentage decrease in total cost over 100 generations for the best solution in each generation. The table is colour coded with a heatmap, with values in green indicating larger percentage decreases and values in red indicating smaller percentage decreases.

TABLE 5-5. Percentage decrease in total cost over 100 generations for different cost weight combinations.

		Cutting Cost Weight				
		0.01	0.1	1	10	100
Packing Cost Weight	1,000	15.15	17.1	9.31	2.78	1.76
	10,000	17.98	10.9	19.35	6.52	1.99
	100,000	4.4	10.59	11.5	24.32	8.7
	1,000,000	0.53	4.54	13.92	15.71	12.61
	10,000,000	0.05	0.4	3.55	7.67	5.75

For the low packing cost/high cutting cost weight results (the 3 red results in the top right corner of Table 5-5), the algorithm struggles to make significant cost savings since the cutting cost term dominates the total cost in the cost function. In such cases, the algorithm cannot add cuts without significantly degrading total cost (even if doing so would improve packing, since the savings in packing would be far outweighed by the increase in cutting cost). Upon closer inspection of these three solutions, it was noted that in all three cases, the number of cuts remained constant (at 6) throughout the optimisation process, confirming this observation. It was however also noted that, despite the number of cuts not changing, in all three cases, the number of containers decreased. For the cutting/packing cost weight combination of (10 / 1000), the number of containers decreased by 3 (14

containers to 11) over 100 generations. For the (100 / 1000) and (100 / 10,000) cost weight cases, the number of containers decreased by 2 (13 containers to 11). This shows that, despite the algorithm being unable to add more cuts (due to large increases in cutting cost), it can still move the existing cuts around to produce cut parts which pack better.

For the high packing cost/low cutting cost weight results (3 red results in the bottom left corner of Table 5-5), an opposite trend was observed. In these cases, the algorithm struggles to make significant cost reductions due to the packing cost term dominating the total cost in the cost function. As such, the algorithm is unable to reduce the cutting without degrading the quality of the packing solution, which would significantly degrade the total cost. Examining the three solutions closer showed that in all three cases, the number of cuts was reduced, but the number of containers remained constant throughout (at 3). For the cutting/packing cost weight combination of (0.1 / 10,000,000), the number of cuts decreased by 7 (36 cuts to 29), and for both the (0.01 / 10,000,000) and (0.1 / 1,000,000) cases, the number of cuts decreased by 8 (36 cuts to 28).

These cases demonstrate that, even under extreme cost weight combinations, the algorithm can still improve the cutting/packing solutions, however the improvement becomes less noticeable (when examining the percentage decrease in total cost) due to one of the cost terms in the cost function dominating the total cost.

Fig. 5-31 shows the cutting/packing solutions for the two most extreme cutting/packing cost weight scenarios; (100 / 1000), shown left, and (0.01 / 10,000,000), shown right, which corresponds to the top right and bottom left cells in Table 5-5 respectively. These examples help illustrate how varying the cost weights can dramatically affect the algorithms behaviour, with high cutting cost weights favouring a reduction in cutting at the expense of more containers (left example), and high packing cost weights favouring a reduction in the number of containers, at the expense of more cutting (right example).

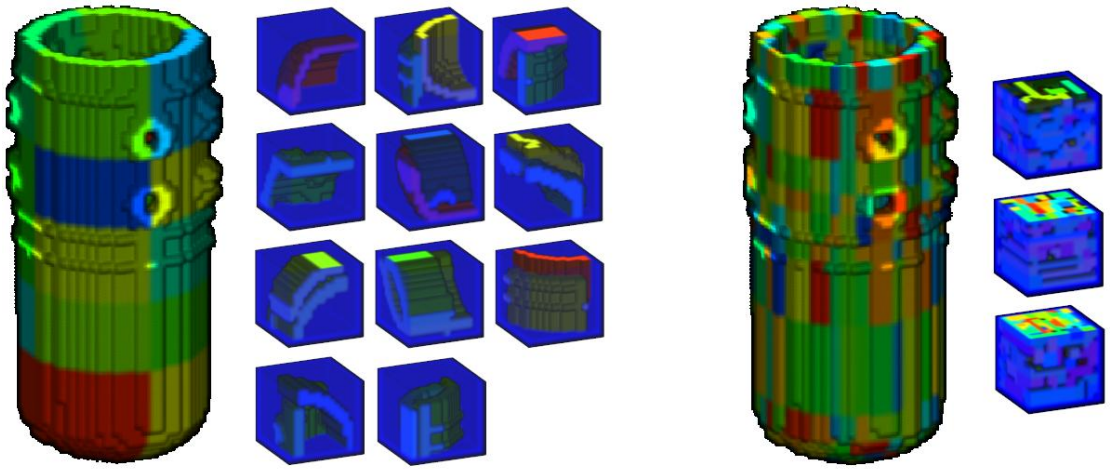


Fig. 5-31. Cutting/packing solutions from extreme cost weight cases; (100 / 1000), shown left, and (0.01 / 10,000,000), shown right.

Additional observations were also made when examining the solutions along the diagonal (top left to bottom right cases). Table 5-6 below shows the solutions along the diagonal from Table 5-5, from top left (0.01 / 1000) to bottom right (100 / 10,000,000), along with the percentage decrease in total cost and the number of cuts and containers in the best initial and best final solutions.

TABLE 5-6. Comparison of initial and final solutions for cases along the diagonal in Table 5-5.

Cutting/Packing Cost Weights	Cost Decrease (%)	Num. Cuts (Initial)	Num. Containers (Initial)	Num. Cuts (Final)	Num. Containers (Final)
0.01 / 1,000	15.15	10	7	9	6
0.1 / 10,000	10.9	13	5	9	6
1 / 100,000	11.5	13	5	9	6
10 / 1,000,000	15.71	10	7	9	6
100 / 10,000,000	5.75	11	6	10	6

From Table 5-6, it can be seen that, whilst the percentage cost decreases along the diagonal are different (column 2), the final solutions (in terms of the number of cuts and containers) are very similar (column 5 and 6). This suggests that when the ratio of the cost weights is the same, the algorithm will converge to similar solutions. This in turn suggests that it is the ratio of the weights, rather than the magnitude of each weight, which has a larger effect on the algorithm's performance.

Given that the percentage cost decreases are different across the diagonal (despite the final solutions being very similar), this suggest that the difference in percentage decrease in cost can be attributed to the quality of the initial solutions found by the algorithm, rather than the weights used. This observation is confirmed by looking at columns 3 and 4 in Table 5-6,

which shows the number of cuts and containers in the best initial solution from the GA runs. Given that the number of cuts and containers is consistent for the final solutions, but not for the initial ones, this verifies that the difference in percentage cost decreases is due to the quality of the initial solutions, with larger percentage decreases being due to poorer initial solutions.

As a final point to note, the highest percentage decrease observed in Table 5-5 (24.32 which corresponds to cutting/packing weights of 10 and 100,000) can also be explained by this phenomenon. Table 5-7 below shows the results (% decrease in cost, number of cuts/containers in best initial and final solutions) for the 24.32 case in Table 5-5 and the two cases diagonally adjacent to it.

TABLE 5-7. Comparison of initial and final solutions between the 24.32 case and the cases diagonally adjacent to it in Table 5-5.

Cutting/Packing Cost Weights	Cost Decrease (%)	Num. Cuts (Initial)	Num. Containers (Initial)	Num. Cuts (Final)	Num. Containers (Final)
1 / 10,000	19.35	9	10	7	8
10 / 100,000	24.32	9	10	7	7
100 / 1,000,000	12.61	9	9	7	8

When comparing the 19.35 and 24.32 cases, the number of cuts and containers in the best initial solutions are the same (columns 3 and 4). However, for the best final solutions, whilst the number of cuts is still the same across both solutions, the number of containers is worse for the 19.35 case compared to the 24.32 case (columns 5 and 6). As such, the reason for this larger decrease in cost for the 24.32 case can be partly attributed to the fact that it converged to a better solution (with fewer containers).

Additionally, when comparing the 24.32 case to the 12.61 case, the number of cuts in the best initial solutions is the same for both cases, but the number of containers for the 12.61 case is better. In contrast, for the best final solutions, the number of cuts is still the same for both cases, but the number of containers is better for the 24.32 case.

From these observations, two conclusions can be drawn:

1. For the 24.32 case, the reason for such a large decrease in total cost is due to the algorithm starting with a poor solution and converging to a very good one (i.e. the difference between the best initial and final solutions is larger, leading to a larger percentage decrease in cost).

2. In contrast, for the 12.61 case, the reason for the smaller decrease in cost when compared to the other two cases in Table 5-7 is because it started with a good solution (better than for the other two cases), meaning the difference between the best initial and final solutions is smaller (leading to a smaller percentage decrease in total cost).

5.6 Chapter 5 Summary

In this chapter, the updated methodology for the 3D implementation of the cutting and packing algorithm has been presented (Section 5.1), along with several studies aimed at assessing the performance and effectiveness of the proposed algorithm (Section 5.5). From the studies presented, several conclusions can be drawn:

Three Decommissioning Cases: In the analysis of the three decommissioning cases, it was demonstrated that the proposed algorithm can consistently achieve significant reductions in cost when compared to a separate 'minimal cutting followed by packing' approach. Additionally, the algorithm was shown to have good performance across a range of structure complexities, achieving average cost reductions between 15.1% and 16.9%. From the analysis of the total cost plots, it was shown that algorithm converges rapidly, with 95% of attainable cost savings being achieved within the first 0-20 generations, and 99% within approximately 20-60 generations. The algorithm also showed high consistency, with the average deviation in total cost across the 5 runs for each structure being approximately $\pm$2-3% of the final mean cost across the 5 runs.

The analysis of the individual cutting and packing costs showed that the proposed methodology can make trade-offs between the two conflicting costs to drive down total cost. It was also noted however that the packing cost converges early in the optimisation process, highlighting the potential for improvements through the use of more advanced packing methods or post-processing steps to improve packing further.

The evaluation of the initialisation strategy showed that, whilst the fixed cutting approaches initialised with 'small', 'medium' and 'large' size constraints can provide good starting points for the algorithm, random initialisation occasionally found better starting solutions. This suggests a need for a more advanced initialisation strategy to help enhance diversity and solution quality further.

Examining the best solutions produced by the algorithm for each structure provided further insights into the efficacy of the chosen cutting and packing approaches. The orthogonal

cutting and PBP packing approaches showed strong performance for the glovebox and RPV, producing regular-shaped parts which facilitated good packing, leading to good container utilisation. However, the heat exchanger highlighted limitations with the existing methods. The complex geometry of the heat exchanger resulted in suboptimal separation of pipe segments from the partitioning baffles which in turn negatively affected the packing algorithm's ability to efficiently pack the parts (particularly for the later containers).

Benchmarking Against Manual Approach: The comparison between the manual trials and the GA highlighted the competitiveness of the proposed approach, with the best results from the manual trials all being within approximately $\pm$2.5% of the mean cost from the 5 20-population GA runs for each structure. The single 100-population run of the GA on each structure demonstrated that with a larger population size, the GA can outperform the human user (with cost reduction of between 0.05-3.38% compared to the best manual results). Although computationally expensive (especially with a larger population) this demonstrates that the algorithm can produce solutions of equivalent quality (and in some cases better) when compared to the manual approach.

The analysis of the user strategy helped provide further insight into the approach adopted by the user when tackling this task, highlighting the tendency to place certain 'feature-based' cuts which remained largely unchanged throughout the manual trials due to their logical geometric placement. This suggests that implementing a feature-based partitioning scheme to pre-partition the structure with immovable, feature-based cuts could be used to enhance performance by reducing the number of cuts requiring optimisation.

Cost Weight Sensitivity Analysis: The analysis of the algorithm under different cost-weight combinations demonstrated how varying the cost weights to favour cutting or packing can dramatically affect the final solutions produced by the algorithm. It also showed that even under extreme cost weight scenarios (where one cost term dominates the total cost), the algorithm is still able to find improvements. Notably, the study also revealed that final solutions produced largely depend on the ratio of the cost weights rather than their absolute magnitudes, and showed that the quality of the initial and final solutions will have a large impact on the achievable percentage decrease in cost.

5.6.1 Future Work

Based on the observations from these studies, there are several areas for potential future work:

Improvements to the GA

Given the observations that for all three decommissioning cases, 99% of attainable cost was achieved within approximately 20-60 generations, a logical step to improve the algorithm would be to implement a stopping criteria based on the convergence, rather than using a fixed number of generations. This could include, for example, keeping track of the best-found solution so far and stopping the algorithm if the improvement in the best solution is less than 1% after a set number of generations. The benefit of this is that it could cut down the run times for the algorithm, whilst still producing results almost identical in terms of total cost to the solutions produced after 100 generations.

Given that the algorithm managed to perform well with a small population (with only marginal gains found with a large one), another potential avenue for future work could be to adapt the GA into a 'Micro-GA'. Micro-GA's, originally proposed in [213], are small population genetic algorithms specifically designed for problems where evaluation of the objective function is computationally expensive. They typically work with a very small population (potentially as low as 4 individuals [214]) of high quality solutions, and incorporate diversity preservation mechanisms (such as restarting the population with random individuals whilst retaining the best previous solution [215]) to help maintain diversity throughout the search. Micro-GA's, though not as well researched as standard GA's, have proven themselves to be effective at solving a wide range computationally intense problems, often being capable of performing as well as, or even better than standard GA's [215-219].

To further enhance the search abilities of the algorithm, a self-adapting mechanism could also be incorporated into the GA to allow the hyperparameters to be dynamically adjusted throughout the search process. It is well known in GA literature that choice of parameters will have a large impact on performance and that parameters must carefully be tuned depending on the problem to be solved [220-223]. One of the limitations with the work presented in this study is that, due to time constraints, the effects of parameter choices could not be explored in detail across a range of structures. The problem with this is that the parameters which were tuned for the simple RPV/container combination used in Section 5.3, may not generalise across all structures (i.e. a set of parameters which is optimal for one structure, may not be optimal for another).

The main principle behind a self-adapting GA is to allow the hyperparameters (such as the mutation rate, population size or tournament size) to dynamically change as the search progresses, thereby encouraging better exploration of the search space and reducing the

likelihood of premature convergence to poor solutions. It has been shown in literature that such self-adapting GA's show great promise when compared to their fixed hyperparameter counterparts, often being able to outperform them due to their better ability to effectively balance exploration and exploitation throughout the search [224-228].

The benefits of incorporating a self-adapting mechanism into the cutting and packing GA are twofold. Firstly, it removes the need for parameter tuning, helping to reduce setup times and allowing the hyperparameters to automatically adapt to any input structure. Secondly, by allowing the parameters to dynamically adjust in response to evolving population characteristics (such as the population diversity), it can help further enhance the search process to allow for better solutions to be found.

Improvements to Packing

Given the fast convergence of the packing costs in the three decommissioning scenarios and the observation of greedy behaviour when packing the heat exchanger parts, another logical step for future work would be to seek ways to enhance the packing optimisation.

Regarding the greedy behaviour observed for the heat exchanger packing, one possible solution could be to try and improve the allocation objective function to try and further reduce the greediness in part selection. The problem, however, is that this may be insufficient, since the allocation process has no awareness of how well an allocated subset will pack in practice. A subset that appears optimal in terms of volume usage may still result in inefficient packing. For example, when comparing the solutions from the heat exchanger and RPV, the heat exchanger had a broader distribution of parts but still showed greedy packing (leading to poorly utilised containers). In contrast, for the RPV, despite the parts having relative uniformity (i.e. poor volume distribution), they packed together effectively due to their block-like geometry. This shows that volume distribution alone is a poor indicator of packability.

A potential solution to improve packing then, could be to introduce a post-processing step which reviews the final packing solution, identifies underutilised containers, and then selectively re-cuts objects to redistribute them more effectively (with the aim of reducing the number of containers). For example, Fig. 5-32 below shows the final 4 containers from the best heat exchanger cutting and packing solution (originally shown in Fig. 5-26). Using the proposed post processing step, the algorithm could take the objects in the last container (U% = 0.176) and try to cut them smaller and repack them into the previous containers.

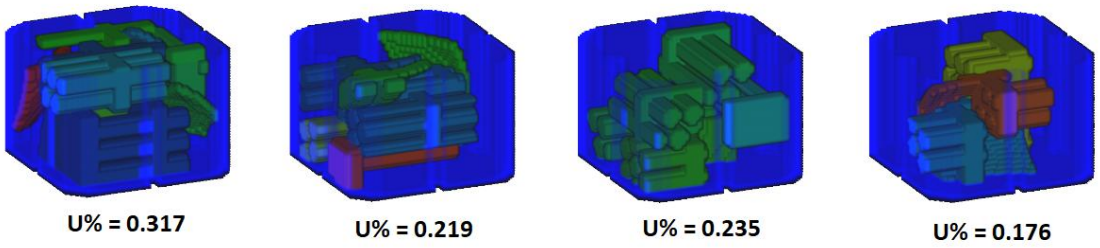


Fig. 5-32. Example of poorly utilised containers from the best heat exchanger run.

Crucially, such a post processing move would only be accepted if it results in a reduction in total cost, i.e. if the savings in packing cost (from a reduction in the number of containers) outweighs the increase in cutting cost (from the additional cutting of parts).

One issue with this post processing step, however, is that it would increase computational overhead if run in the main optimisation loop. To alleviate this problem, two possible strategies could be considered:

1. Apply post-processing selectively to a small number of top-ranked solutions in each generation within the cutting and packing GA.
2. Run post-processing only on the final best solution, outside the GA loop.

Another logical step to improve packing would be to improve the placement process itself (i.e. the DigiPac placement strategy). One possible way to do this would be to implement the updated hyper-heuristic framework proposed in Chapter 4, Section 4.2.4. By using an intelligent selection strategy (which considers the geometry of both the part to pack and the packed pile) for selecting which heuristic to use for placing a part, it would remove the need for metaheuristic optimisation of placement heuristics (reducing computational overhead) whilst still giving the algorithm greater flexibility in terms of where it can place parts (thereby improving utilisation).

Another potential improvement to the placement process would involve implementing the methodology used to find placement sites as proposed in [229]. The problem with the placement process utilised by DigiPac is that it requires scanning the object across the discretised packing space voxel by voxel, checking for collisions at each step. The problem with this is that it becomes computationally expensive as the size of the grid increases. In contrast, the approach in [229] (referred to as ‘spectral packing’) reformulates the collision detection and site evaluation as convolution operations performed in the frequency domain using Fast Fourier Transform (FFT). This allows collision-free positions to be identified across the entire container volume in a single batch operation (i.e. all at once), as opposed to sequentially (as in the scanning process used by DigiPac), greatly reducing

computational complexity. For example, they claim that using a brute force scanning process on a 240x123x100 voxel grid takes 9.4 seconds, while their FFT approach reduces this time down to 0.003 seconds. If true, adopting such an approach could potentially increase the speed of packing by several orders of magnitude, allowing for higher resolution, large scale packing problems to be tackled more efficiently.

As a final point to note, more could also be done to improve the real-world applicability of the packing algorithm by incorporating the following:

- **Radiation and weight-aware allocation**: As discussed in the literature review, using hard limits for radiation and weight on the container runs the risk of poor container utilisation if one of the limits is hit first. E.g. if a small number of highly radioactive components are allocated to a single container, the radiation limit could be reached before the weight limit or container capacity is reached, resulting in poor utilisation. In contrast, using an allocation strategy which specifically optimises for these objectives during allocation could re-distribute these small radioactive parts across multiple containers, allowing more to be packed into them.
- **Incorporating stability checking:** Whilst the stability of packed parts wasn't considered in this project, it will be essential to consider in future to ensure packing structure can be recreated in the real world (i.e. to ensure that parts don't shift unexpectedly during packing, preventing the structure from being reproducible). Whilst DigiPac currently has the ability to simulate dynamic forces during packing using its in-built DEM (Discrete Element Method) packing process, the problem is that it is very computationally expensive. As such, it would be impractical to use this packing approach in the GA optimisation loop. Future work should focus on ways to incorporate fast stability checking that can be integrated into the optimisation process with minimal computational overhead.
- **Waste stream segregation:** In real-world decommissioning, waste is often sorted into different categories (such as low-, intermediate- and high-level waste) depending on radioactivity, with each category often requiring different containers and shielding levels. Another improvement could be to incorporate radiation information for the cut parts and allow the user to specify different containers for the different classifications. In doing so, the packing algorithm could separate parts into different waste streams and pack them into the different containers depending on which waste stream they belong to.

Improvements to Cutting

One promising avenue for future works stems from the observation made during the manual benchmarking that human users tend to place certain 'feature-based' cuts (e.g. to separate the domed cap from the main body of the heat exchanger, or separating the flat sides of the glovebox), which remain largely unchanged throughout subsequent modifications to the cutting patterns.

One way to achieve this would be to use 'semantic segmentation', which is a technique commonly used in computer vision and 3D shape recognition to assign class labels or part categories to regions, or subcomponents, of a 3D object [230]. This would involve using point cloud or voxel-based deep-learning models to segment structures into semantically meaningful components, such as 'side wall', 'end cap', 'partition baffle', 'pipe segment', etc. Algorithms such as the ones developed in [230-234] have shown considerable success in this domain, particularly for indoor scenes and object decomposition (Fig. 5-33). Such models could potentially be repurposed to segment decommissioning structures into meaningful subcomponents.

Once segmented, the algorithm could then add cuts between major subcomponents as a pre-processing step, leaving the cutting and packing optimisation process to focus on optimising cuts on the subcomponents. This would reduce the number of optimisation variables, allowing the algorithm to focus the search on more uncertain cuts (cuts where the effect on solution quality is more uncertain).

The primary challenge with such an approach would be in training an AI model to reliably recognise these structural features. One solution would be to train the model using manually segmented and labelled structures; however, this would require significant time and manual effort. Therefore, future work should focus on developing approaches that can learn to identify key structural components with limited human supervision.

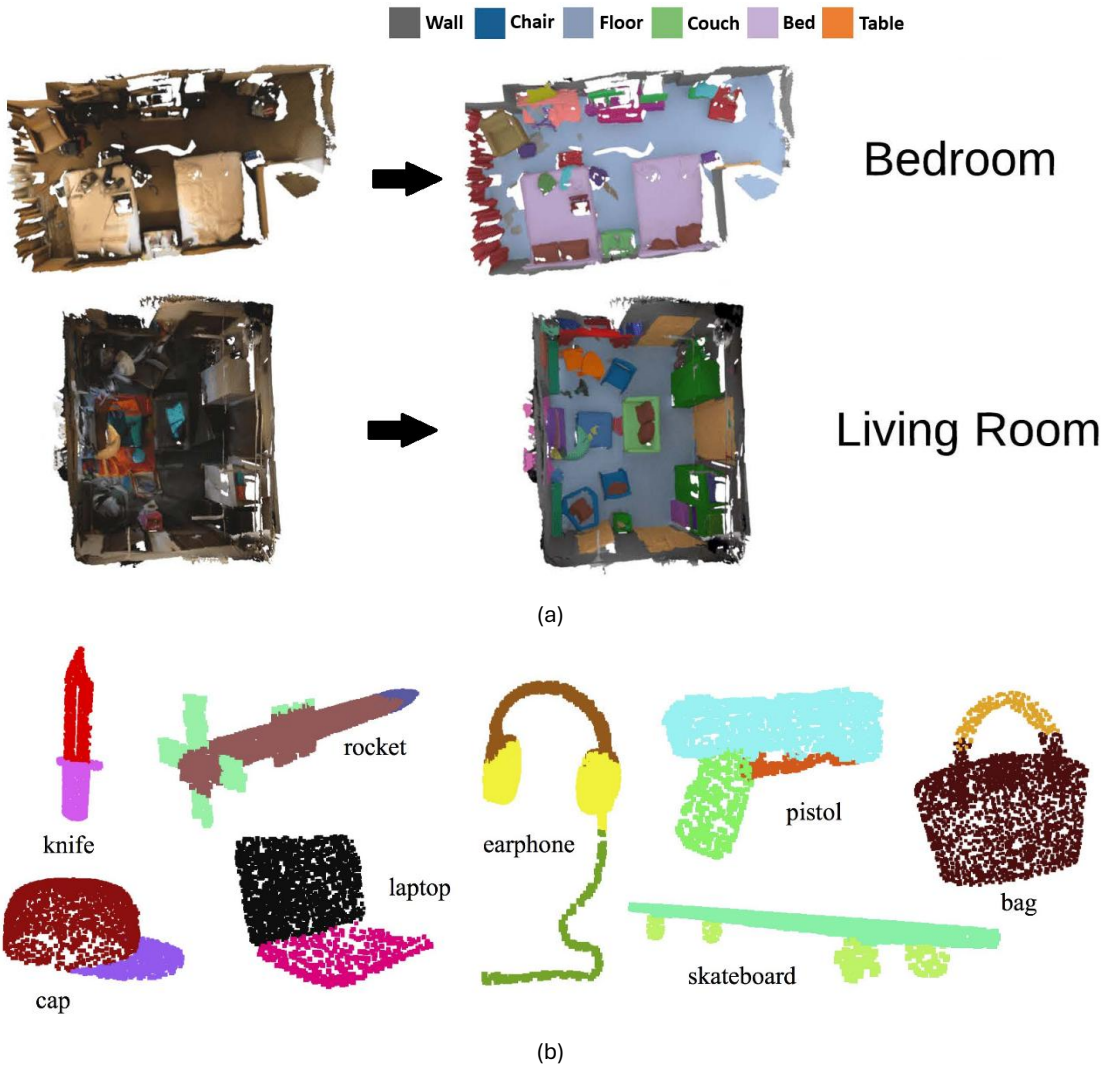


Fig. 5-33. Examples of semantic segmentation. (a) – Segmentation of RGBD scanned rooms, from [233]. (b) – Part segmentation of point cloud scans, from [231].

In addition to this, more could also be done to improve the real-world applicability of the cutting process, such as:

- **Better cost estimation:** With the current implementation, it is assumed that the same cutting tool is used for the entire structure (i.e. a single cutting cost weight in the cost function, and a fixed cut width applied to all cuts). As discussed in the literature review (Chapter 2, Section 2.3), it would be beneficial to incorporate support for multiple cutting tools (with different cost weights and cut widths) depending on the material being cut through. For example, this could involve creating a user interface to allow the user to label different regions of the structure

and assign different cost weights/cut widths to them. Going further, if a segmentation algorithm is also incorporated, an AI model could be trained to recognise such regions and assign appropriate cutting parameters automatically (removing the need for manual labelling).

- **Region avoidance:** In the current implementation, cuts are allowed to pass through any part of the structure. However, as discussed in the literature review (Chapter 2, Section 2.3), certain regions may need to be preserved or avoided. These could include, for example, areas of high radiation (which may be safer to isolate and remove as a single piece to minimise ambient exposure), or parts of a structure which may be difficult or impractical to cut through (such as a pump housing connected to the side of a tank). It would therefore be beneficial to allow such parts of a structure to be identified and labelled (either manually or by a trained AI) prior to optimisation. Once identified, these regions could be excluded from cutting by either isolating and removing them prior to optimisation or by incorporating an avoidance mechanism into the algorithm (e.g. penalising cuts that intersect protected zones or enabling cuts to deform around them).
- **Disassembly sequencing:** As demonstrated in the 2D investigation (Chapter 4, Section 4.4), disassembly sequencing is an important aspect of real-world applicability (especially if the goal is to perform cutting and packing using autonomous robotic systems). In the current 3D algorithm, disassembly sequencing was not considered, meaning that cutting patterns could theoretically result in obstructed or unstable removal sequences. Future work should focus on integrating disassembly considerations into the optimisation loop (by incorporating fast stability checks and collision-free path planning for cut parts). Building on the 2D work, the aim would be to develop a fast and robust disassembly planner capable of validating whether cutting patterns are physically realisable during deconstruction.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

The research presented in this thesis set out to address a fundamental challenge in the decommissioning of large, irregular structures: how to plan the cutting and packing of a solid structure in a way that balances the trade-off between size-reduction effort and packing efficiency. This challenge is particularly relevant in nuclear decommissioning, where the physical dismantling and packaging of large structures must be carefully optimised to minimise total cost and ensure regulatory compliance.

Due to the absence of integrated approaches that can jointly optimise both cutting and packing, this project aimed to develop a computational optimisation framework capable of generating cost-effective cutting and packing plans for irregular 3D structures. Specifically, the objective was to create an algorithm that could take a 3D input model of a structure and produce a cost-optimised cutting plan and corresponding packing configuration, whilst accounting for real-world constraints such as tool usage and container type.

To achieve this, the work was conducted in several phases: a literature review to identify limitations with existing approaches to cutting and packing optimisation, the development of a simplified 2D prototype to test proposed concepts, and the creation of a full 3D optimisation framework. A series of tests was performed to evaluate the methodology, including testing the algorithm's ability to handle arbitrary structures of different geometric complexity, benchmarking against conventional and manual methods, and sensitivity analysis for different cost weights.

This chapter concludes the thesis by summarising the main contributions and findings of the research, followed by a consolidated overview of suggested directions for future work.

The first part (Section 6.1) provides a reflection on the key outcomes of this project in the context of the research aims and objectives introduced in Chapter 1. It revisits the challenges addressed in this thesis and highlights how the proposed methodology and experimental testing contribute towards addressing them.

The second part (Section 6.2) outlines future work directions, based on detailed discussions provided throughout the thesis. Rather than repeating in-depth discussions offered in previous sections of the thesis, this section offers a concise summary of opportunities for extending and improving the methodology, grouped around core aspects of the algorithm design.

6.1 Summary of work

This section summarises the work completed in this thesis, structured around the objectives outlined in Chapter 1.

Objectives 1 & 2 – Review existing literature, propose methodology, and evaluate in 2D

The first stage of the research focused on reviewing existing literature on cutting and packing optimisation, with the goal of identifying key limitations and developing a methodology to address them. A detailed literature review was conducted across three main areas: previous efforts to link cutting and packing, packing optimisation for 3D irregular objects, and cutting optimisation within the context of nuclear decommissioning.

In the review on previous attempts at linking cutting and packing (Chapter 2, Section 2.1), a number of limitations were identified. The primary limitation with most existing approaches is that they are highly sensitive to initialisation (i.e. they rely heavily on the quality of the initial partitioning), with limited ability to modify the cutting patterns during the optimisation process. Additionally, the strategies often fail to re-optimise packing from scratch each time the cutting pattern is modified (meaning the algorithm may potentially miss good cutting/packing solutions). Ultimately, this means the feedback loop between cutting and packing is either weak or absent, preventing these algorithms from accurately capturing the trade-off between segmentation effort and packing efficiency. Furthermore, it was also noted that these algorithms all assume the cut parts will fit into a single container, with no ability to pack objects across multiple containers (restricting their applicability to real-world problems where a single container is rarely sufficient).

Given that within cutting and packing optimisation, packing is the limiting factor (both from a computational and cost perspective), a comprehensive review on irregular object packing in 3D was also presented (Chapter 2, Section 2.2). This review helped identify commonly used placement strategies and metaheuristic algorithms used in packing optimisation, alongside presenting a detailed discussion on key factors relating to packing algorithm design. Several research gaps were also identified, including the need for consistent benchmarking datasets, limited research into different multi-container packing strategies, and the incorporation of higher-level decision strategies such as hyper-heuristics.

The final part of the review focused on cutting optimisation algorithm design within the context of nuclear decommissioning (Chapter 2, Section 2.3). This section identified several practical considerations that are often overlooked in cutting optimisation algorithm

design (including the influence of different cutting tools on cost, the importance of minimising radiation exposure, and the need for feasible part removal), highlighting several areas for potential future work

Based on these findings, a new methodology was proposed (Chapter 3, Section 3.2), in which cutting and packing are treated as a coupled optimisation problem. The core concept is a feedback driven approach in which a genetic algorithm (GA) generates candidate cutting patterns. Each pattern is evaluated by the cutting and packing sub-processes, and both the cutting and packing costs are then used to evaluate the quality of the cutting patterns and guide their modification. By using a population-based optimisation approach (GA) with tailored genetic operators for crossover and mutation, the proposed method alleviates the problems with previous approaches by reducing the reliance on a good initial partitioning and allowing cuts to be modified in a more flexible manner during optimisation.

To test the feasibility of the proposed approach, a simplified 2D implementation of the algorithm was developed (Chapter 4, Section 4.1). This prototype incorporated a basic cutting strategy and a container packing algorithm, with both processes embedded within a genetic algorithm optimisation loop. The performance of the proposed method was benchmarked against a random search strategy that modified cutting patterns without guidance. The results showed that the feedback-driven GA outperformed the random approach, demonstrating that integrated optimisation of cutting and packing can produce higher-quality solutions.

However, the 2D implementation also revealed several challenges, particularly with the chosen representation of cutting patterns and the use of penalty functions for dealing with infeasible solutions (cut parts larger than the container). The binary space partitioning (BSP) approach used in the initial implementation required multiple parameters per cut (leading to a high number of variables), and the use of penalty functions resulted in the algorithm wasting excessive time evaluating infeasible (and poor quality) solutions. Both factors resulted in slow convergence, leading to several key design changes being made for the 3D algorithm (including the adoption of a simplified orthogonal cutting approach and a feasibility-preserving repair mechanism).

Objective 3 - investigate ways to enhance solutions quality and applicability

As noted in the literature review, there are many research gaps which exist within the topics of cutting and packing optimisation. Due to time constraints, only a subset of these

problems were explored in this project. To simplify implementation and accelerate testing, the studies were conducted using 2D representations (with all methodology being transferable to 3D). These studies (originally presented in Chapter 4) were designed to explore methods for enhancing the solution quality and real-world applicability of the cutting and packing processes within the broader optimisation framework. The studies investigated were as follows:

- **Chapter 4, Section 4.2- Hyper-heuristics for packing optimisation:** Given the limited attention given to hyper-heuristics in packing literature and the use of a placement heuristic-based packing approach in this project, hyper-heuristics were identified as a natural extension for improving solution quality in packing. A novel hyper heuristic framework was proposed which used a genetic algorithm to optimise the placement heuristic (from a choice of four) used to pack each object. After successfully demonstrating the efficacy of the proposed framework for single container packing, the methodology was extended to multi-container packing by combining placement heuristic optimisation with packing order optimisation (to change the allocation of objects to different containers). Testing showed that the hyper-heuristic approach can outperform literature, especially for small to moderate object sets. However, performance declined with an increasing number of objects due to the factorial growth in permutation of packing orders, highlighting scalability issues.
- **Chapter 4, Section 4.3- Comparison of multi container packing strategies:** Following the scalability limitations observed with the order-optimisation based hyper-heuristic, an alternative multi-container packing strategy (called Partial Bin Packing – PBP) was investigated. Unlike the hyper-heuristic algorithm, where the allocation of objects to different containers is tied to the packing order, PBP separates the allocation process from the packing process and optimises the packing one container at a time. It works by using a separate allocation algorithm to select subsets of object to allocate to a container, before the objects are packed by the packing algorithm. This decoupling of the allocation and packing reduces computational burden, enabling solutions to be generated much faster. Comparative testing showed that PBP consistently outperformed the hyper-heuristic method in terms of speed and packing quality for large object sets, making it better suited for real-world decommissioning problems involving dozens or hundreds of cut parts.

- **Chapter 4, Section 4.4 - Disassembly sequencing for cut structures:** A third study was conducted to explore disassembly sequencing for enhancing the realism of the cutting process. This was motivated by the observation that a viable cutting plan must not only produce efficiently packed parts but also ensure that those parts can be physically removed from the structure without obstruction or instability. Four optimisation strategies were implemented and compared for finding feasible disassembly sequences based on a simplified 2D representation of a cut structure, incorporating constraints for mechanical stability and part accessibility. Whilst this methodology could not be implemented into the 3D cutting and packing algorithm due to time constraints, the study demonstrated its potential value by highlighting how different algorithms varied in their ability to find feasible and cost-effective disassembly sequences. The study revealed key trade-offs between computational efficiency, sequence feasibility, and solution optimality, with some strategies able to quickly find valid sequences, while others were more prone to becoming trapped in infeasible dead-ends, or required significantly more time to explore the solution space.

Objectives 4 & 5 – Extend the framework to 3D and evaluate performance

The final stage of the project focused on extending the cutting and packing framework to 3D and evaluating its performance across several case studies (Chapter 5). Based on the insights gained from the 2D studies, several key modifications were made to the methodology, including replacing the original BSP cutting approach with a simplified orthogonal cutting strategy, introducing a repair mechanism to ensure cut parts remained within the packing constraints, and implementing the PBP approach for fast multi-container packing (Chapter 5, Section 5.1).

The 3D algorithm was tested on three case studies, each representing different structures of varying geometric complexity: a glovebox, a reactor pressure vessel (RPV), and a heat exchanger. For each case, the performance of the algorithm was evaluated in terms of total cost reduction, convergence rate and execution time. In addition, results were benchmarked against both a minimal cutting strategy (where segmentation is minimised and only packing is optimised) and a set of 30 manually designed cutting plans created by a human user.

The results showed that the proposed integrated cutting and packing approach consistently outperformed the minimal-cutting-followed-by-packing approach, achieving average total

cost reductions between 15-17%. The GA optimisation process also showed strong convergence behaviour, with 95% of attainable cost savings achieved within the first 20 generations and 99% by generation 60. Additionally, the algorithm was found to be highly consistent, with a typical variation of only ±2-3% in the final total cost across repeated runs.

Benchmarking against the manual approach showed that the algorithm can produce solutions of comparable (and sometimes even superior) quality. With a larger population size (of 100 compared to 20), the algorithm achieved up to 3.38% improvement in total cost compared to the best manual solutions, demonstrating that this automated approach can rival expert human judgement.

Analysis of the final cutting and packing results also offered insights into the strengths and limitations of the methodology. The orthogonal cutting and PBP packing approach showed strong performance for the more regular-shaped structures like the glovebox and RPV, but struggled with the more complex geometry of the heat exchanger, where packing efficiency was more sensitive to the separation of parts. These findings reinforce the robustness of the approach while also highlighting opportunities for future improvement, particularly in handling highly complex geometries and improving container utilisation.

6.2 Future work

This section consolidates the various opportunities for extending and enhancing the cutting and packing framework, as discussed throughout the thesis. Suggested directions for future work are grouped into three categories: algorithmic improvements (improvements to the GA process used to link cutting and packing), packing enhancements (improvements to the underlying packing process), and cutting enhancements (improvements to the underlying cutting process).

6.2.1 Algorithm Improvements

This subsection outlines general improvements to the GA optimisation strategy, motivated primarily by observations made during testing of the 3D algorithm (Chapter 5).

- **Improved stopping criteria:** As discussed in Chapter 5, Section 5.6.1, the algorithm currently uses a fixed number of generations as its termination condition. Whilst this simplifies implementation, it can lead to unnecessary computation once the search plateaus (recall that by generation 60, 99% of attainable cost savings had

been achieved). Introducing dynamic stopping criteria (such as stopping the search when the improvement in total cost across a small window of generations falls below a threshold), would allow the algorithm to terminate early when convergence has occurred, saving time with minimal impact on solution quality.

- **Micro-GA:** As shown in Chapter 5, Section 5.3.3, the algorithm showed strong performance even with a relatively small population size (of 20), with only marginal gains observed from using a larger one (of 100). As discussed in Chapter 5, Section 5.6.1, this suggests that adapting the GA to a 'Micro-GA' ([215-219]) could be a promising direction for future work. Micro-GAs are a specialised type of GA designed specifically for problems where evaluating the objective function is computationally expensive (as in this project). They operate with a very small population of high-quality solutions and employ diversity preserving mechanisms to prevent premature convergence. Though not as widely studied as traditional GAs, they have shown strong performance across a range of computationally intensive optimisation tasks. Adopting a Micro-GA could significantly reduce run-times while maintaining solution quality.
- **Self-Adapting GAs:** The GA used in this project relies on manually tuned parameters (see Chapter 3, Sections 3.3.4 and 3.3.5). As discussed in Chapter 5, Section 5.6.1, allowing parameters to dynamically change as the search progresses can help the algorithm explore the search space better (e.g. by increasing the mutation rate when population diversity is lost to encourage exploration, or increasing selection pressure in a highly diverse population to encourage exploitation of fitter individuals). Future work should also seek to implement self-adapting strategies to allow the parameters to dynamically change in response to evolving population characteristics (e.g. diversity).

6.2.2 Improvements to Packing

This subsection outlines improvements to the packing side of the algorithm, based on limitations identified during testing of the 3D algorithm (Chapter 5) and earlier 2D investigations (Chapter 4).

- **Hyper-Heuristics for packing:** One of the main limitations with the GA-based hyper-heuristic algorithm proposed in Chapter 4, Section 4.2 was that it required metaheuristic optimisation to trial many different packing order/placement heuristic combinations (leading to long computation times). In Section 4.2.4, an

updated methodology was proposed where the idea is to use a high-level intelligent selection strategy to select which placement heuristic to use to pack the next object, based on geometric features of the object and the packed pile. Implementing such an approach would remove the need for metaheuristic optimisation, helping to speed up packing whilst still retaining the enhanced flexibility of a hyper-heuristic approach.

- **Post-Processing for packing improvement:** Analysis of packing solutions showed that in certain cases, not all the containers in a multi-container packing solution had high utilisation. A promising area of future work (as discussed in Chapter 5, Section 5.6.1), would be to implement a post-processing step that analyses poorly utilised containers and attempts to re-cut and redistribute objects in them to reduce the number of containers. This could involve selectively cutting large items from under-utilised containers and repacking them into other poorly utilised containers.
- **Incorporating real-world constraints:** As discussed in Chapter 5, Section 5.6.1, there are also several constraints that should be incorporated into the packing process to increase real world applicability. These include: radiation and weight limits on the containers (to ensure regulatory compliance), incorporating physics engines into packing to ensure that parts remain stable in the container as they are added (to ensure structures can be replicated in the real world), and waste stream separation of objects into different container types depending on radioactivity (to better reflect real world decommissioning practice).

6.3.3 Improvements to Cutting

This subsection outlines proposed improvements to the cutting process, based on findings from the 2D and 3D studies (Chapters 4 and 5) and additional insights from the manual trial analysis.

- **Feature-based initialisation:** a key insight from the manual trials (Chapter 5, Section 5.5.2) was that the user tended to place certain cuts in geometrically meaningful locations (e.g. removing the side panels of the glovebox or separating the domed end cap from the main body of the heat exchanger), which remained largely unchanged across repeated trials. This suggests that some cuts are more ‘natural’ due to the object’s inherent geometry. Future work could investigate incorporating a pre-processing feature-based partitioning step to replicate this

behaviour automatically. Semantic segmentation methods (see Chapter 5, Section 5.6.1) could be used to detect high-level structural components, enabling the algorithm to insert immovable, 'feature-based' cuts before optimisation begins. This could reduce the number of optimisation variables and improve convergence by allowing the GA to focus on more uncertain or non-obvious cut locations.

- **Disassembly sequencing for cut structures:** As stated earlier, disassembly sequencing is an important factor which must be considered to ensure cutting plans created by the algorithm can be executed in the real world. Future work should therefore focus on integrating disassembly feasibility checks into the 3D optimisation loop. This will require fast, scalable algorithms to assess part removability under structural and accessibility constraints. A promising solution (as discussed in Chapter 4, Section 4.4.5) is the proposed hybrid greedy algorithm with height-based backtracking, which combines the efficiency of greedy search with a heuristic backtracking mechanism to escape infeasible dead-ends. Additionally, the simplified stability model used in this study should be benchmarked against physics-based methods (e.g. finite element analysis) to evaluate its predictive accuracy. Finally, to progress toward full robotic execution, future work should also work on incorporating more advanced robotic models and motion planning algorithms for assessing cut part extractability.
- **Tool aware cutting and region avoidance:** As outlined in Chapter 2, Section 2.3.1 and discussed further in Chapter 5, Section 5.6.1, there is more which could be done to improve the realism of cutting plans by accounting for practical cutting constraints. Currently, the algorithm assumes that a single cutting tool is used and allows cuts to pass through any part of the structure. In reality, different parts of a structure may require different tools due to variations in material or geometry, and some regions (e.g. highly radioactive zones or complex components like pump housings) may need to be avoided entirely. Future work should focus on developing a tool-aware cost model, where users can assign different cutting tools (with different costs) to specific parts of a structure. In addition, the algorithm should include region avoidance by either isolating sensitive regions and removing them before optimisation or penalising/redirecting cuts that pass through them.

BIBLIOGRAPHY

[1] I. A. E. Agency, ‘Global Status of Decommissioning of Nuclear Installations’, International Atomic Energy Agency, Text, 2023. Accessed: Mar. 28, 2025. [Online]. Available: https://www.iaea.org/publications/15197/global-status-of-decommissioning-of-nuclear-installations

[2] ‘UK Radioactive Waste Inventory 2022’, GOV.UK. Accessed: Mar. 28, 2025. [Online]. Available: https://www.gov.uk/government/publications/uk-radioactive-waste-and-material-inventory-2022/uk-radioactive-waste-inventory-2022

[3] P. E. O. Lainetti and R. de J. Associacao Brasileira de Energia Nuclear, ‘Cutting techniques for facilities dismantling in decommissioning projects’, Associacao Brasileira de Energia Nuclear, Rio de Janeiro, RJ (Brazil), INIS-BR--11049, Jan. 2011. Accessed: May 12, 2025. [Online]. Available: https://inis.iaea.org/records/rav3n-00269

[4] G.-R. Lee, B.-J. Lim, D.-W. Cho, and C.-D. Park, ‘Selection methodology of the optimal cutting technology for dismantling of components in nuclear power plants’, *Annals of Nuclear Energy*, vol. 166, p. 108808, Feb. 2022, doi: 10.1016/j.anucene.2021.108808.

[5] R. Borchardt *et al.*, ‘Remote handling techniques in decommissioning - A report of the NEA Co-operative Programme on Decommissioning (CPD) project’, Radioactive Waste Management Committee - RWMC, Organisation for Economic Co-Operation and Development, Nuclear Energy Agency - OECD/NEA, Le Seine Saint-Germain, 12 boulevard des Iles, F-92130 Issy-les-Moulineaux (France), NEA-RWM-R--2011-2, Sep. 2011. Accessed: May 12, 2025. [Online]. Available: https://inis.iaea.org/records/d7k4b-s5984

[6] ‘Cost Estimation for Decommissioning’, OECD. Accessed: May 12, 2025. [Online]. Available: https://www.oecd.org/en/publications/cost-estimation-for-decommissioning_9789264106864-en.html

[7] ‘Handbook for Nuclear Decommissioning Cost Estimation’, Nuclear Energy Agency (NEA). Accessed: May 12, 2025. [Online]. Available: https://oecd-nea.org/jcms/pl_90297/handbook-for-nuclear-decommissioning-cost-estimation?details=true

[8] Yumpu.com, ‘Solid Radioactive Waste Strategy Report.pdf - UK EPR’, yumpu.com. Accessed: May 12, 2025. [Online]. Available: https://www.yumpu.com/en/document/view/34867607/solid-radioactive-waste-strategy-reportpdf-uk-epr

[9] ‘Pressure vessel segmented at Bohunice’, World Nuclear News. Accessed: May 12, 2025. [Online]. Available: https://world-nuclear-news.org/articles/pressure-vessel-segmented-at-bohunice

[10] ‘Dragon reactor dismantling underway’, GOV.UK. Accessed: May 12, 2025. [Online]. Available: https://www.gov.uk/government/news/dragon-reactor-dismantling-underway

[11] ‘Orano Completes Industry-first Segmentation and Disposal of a U.S. BWR Reactor at Vermont Yankee and in Less Than Four Years on Accelerated Decommissioning Timeline’, usa.orano. Accessed: May 12, 2025. [Online]. Available:

https://www.orano.group/usa/en/our-news/news-releases/2022/december/orano-completes-industry-first-segmentation-and-disposal-of-a-u-s-bwr-reactor-at-vermont-yankee-and-in-less-than-four-years-on-accelerated-decommissioning-timeline

[12] G. Melis and P. Cretskens, 'Dismantling of Obsolete Installations and Glove Boxes', JRC Publications Repository. Accessed: Apr. 02, 2025. [Online]. Available: https://publications.jrc.ec.europa.eu/repository/handle/JRC30955

[13] A. Zorliu and C. Petran, 'Research Reactor Decommissioning Demonstration Project (R^2D^2P) Workshop on Dismantling of the higher active parts of a research reactor', Bucharest-Magurele, Romania, 2015. Accessed: Apr. 02, 2025. [Online]. Available: https://nucleus.iaea.org/sites/connect/IDNpublic/R2D2/Workshop%2013/7%20Dismantling%20technologies%20and%20tools.pdf

[14] X. Jia, 'Simulating the Dismantling/Packing Process using NuPlant', University of Leeds, 2009.

[15] A. Maggini, R. Ciolini, S. Pistelli, and E. Garneri, 'Strategies and dismantling solutions for RPV internals of Trino NPP', *Progress in Nuclear Energy*, vol. 93, pp. 67–75, Nov. 2016, doi: 10.1016/j.pnucene.2016.08.001.

[16] L. Andreas, S. Dieter, and K. Lutz, 'Decommissioning of the reactor pressure vessel and its peripheral facilities of the Nuclear Power Plant in Stade, Germany', presented at the Waste Management Symposia, 2011. [Online]. Available: https://archivedproceedings.econference.io/wmsym/2011/papers/11100.pdf

[17] 'Decommissioning: Reactor Pressure Vessel Internals Segmentation'. EPRI, 2001. Accessed: Apr. 02, 2025. [Online]. Available: https://restservice.epri.com/publicdownload/000000000001003029/0/Product

[18] P.-C. Tsai, X. Huang, Y.-H. Hung, C.-H. Huang, and S. Smith, 'The computer aided cutting planning of components using genetic algorithms for decommissioning of a nuclear reactor', *Annals of Nuclear Energy*, vol. 130, pp. 200–207, Aug. 2019, doi: 10.1016/j.anucene.2019.02.041.

[19] J.-Y. Li *et al.*, 'The meta-heuristic method based cutting planning for the decommissioning of spallation target in China initiative accelerator driven subcritical system', *Annals of Nuclear Energy*, vol. 170, p. 108978, Jun. 2022, doi: 10.1016/j.anucene.2022.108978.

[20] Y. Zhao and C. T. Haas, 'A 3D Irregular Packing Algorithm Using Point Cloud Data', in *Computing in Civil Engineering 2019*, Atlanta, Georgia: American Society of Civil Engineers, Jun. 2019, pp. 201–208. doi: 10.1061/9780784482438.026.

[21] H. C. Kim, S. H. Han, Y. J. Lee, and D. I. Kim, 'Packing placement method using hybrid genetic algorithm for segments of waste components in nuclear reactor decommissioning', *Nuclear Engineering and Technology*, vol. 54, no. 9, pp. 3242–3249, Sep. 2022, doi: 10.1016/j.net.2022.04.004.

[22] M. Yao, Z. Chen, L. Luo, R. Wang, and H. Wang, 'Level-set-based partitioning and packing optimization of a printable model', *ACM Trans. Graph.*, vol. 34, no. 6, pp. 1–11, Nov. 2015, doi: 10.1145/2816795.2818064.

[23] J. Vanek *et al.*, 'PackMerger: A 3D Print Volume Optimizer', *Computer Graphics Forum*, vol. 33, no. 6, pp. 322–332, Sep. 2014, doi: 10.1111/cgf.12353.

[24] X. Chen *et al.*, ‘Dapper: decompose-and-pack for 3D printing’, *ACM Trans. Graph.*, vol. 34, no. 6, pp. 1–12, Nov. 2015, doi: 10.1145/2816795.2818087.

[25] M. Attene, ‘Shapes In a Box: Disassembling 3D Objects for Efficient Packing and Fabrication’, *Computer Graphics Forum*, vol. 34, no. 8, pp. 64–76, Dec. 2015, doi: 10.1111/cgf.12608.

[26] R. Hu, H. Li, H. Zhang, and D. Cohen-Or, ‘Approximate pyramidal shape decomposition’, *ACM Trans. Graph.*, vol. 33, no. 6, p. 213:1-213:12, Nov. 2014, doi: 10.1145/2661229.2661244.

[27] M. Laraia, *Nuclear decommissioning: Planning, execution and international experience*. 2012, p. 824.

[28] W. Tian, Y. Zhang, W. Chen, J. Zhang, B. Ni, and C. Jiang, ‘A Multi-Objective Optimization Design for Enhancing Space Utilization and Minimizing Cutting Time in Nuclear Facility Component Decommissioning’, *Nuclear Science and Engineering*, vol. 199, no. 3, pp. 518–529, Mar. 2025, doi: 10.1080/00295639.2024.2372516.

[29] D. Domovic, T. Rolich, D. Grundler, and S. Bogovic, ‘Algorithms for 2D nesting problem based on the no-fit polygon’, in *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia: IEEE, May 2014, pp. 1094–1099. doi: 10.1109/MIPRO.2014.6859732.

[30] H. Dong, G. M. Fadel, and V. Y. Blouin, ‘Packing Optimization by Enhanced Rubber Band Analogy’, presented at the ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers Digital Collection, Jun. 2008, pp. 681–689. doi: 10.1115/DETC2005-85502.

[31] C. Lamas-Fernandez, J. A. Bennell, and A. Martinez-Sykora, ‘Voxel-Based Solution Approaches to the Three-Dimensional Irregular Packing Problem’, *Operations Research*, vol. 71, no. 4, pp. 1298–1317, Jul. 2023, doi: 10.1287/opre.2022.2260.

[32] J. K. Dickinson and G. K. and Knopf, ‘Packing Subsets of 3D Parts for Layered Manufacturing’, *International Journal of Smart Engineering System Design*, vol. 4, no. 3, pp. 147–161, Jan. 2002, doi: 10.1080/10255810213478.

[33] T. Romanova, J. Bennell, Y. Stoyan, and A. Pankratov, ‘Packing of concave polyhedra with continuous rotations using nonlinear optimisation’, *European Journal of Operational Research*, vol. 268, no. 1, pp. 37–53, Jul. 2018, doi: 10.1016/j.ejor.2018.01.025.

[34] T. Romanova *et al.*, ‘Sparsest balanced packing of irregular 3D objects in a cylindrical container’, *European Journal of Operational Research*, vol. 291, no. 1, pp. 84–100, May 2021, doi: 10.1016/j.ejor.2020.09.021.

[35] N. Chernov, Yu. Stoyan, and T. Romanova, ‘Mathematical model and efficient algorithms for object packing problem’, *Computational Geometry*, vol. 43, no. 5, pp. 535–553, Jul. 2010, doi: 10.1016/j.comgeo.2009.12.003.

[36] I. Litvinchev, A. Pankratov, and T. Romanova, ‘3D Irregular Packing in an Optimized Cuboid Container’, *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 2014–2019, 2019, doi: 10.1016/j.ifacol.2019.11.499.

[37] A. Pankratov, T. Romanova, S. Shekhovtsov, I. Grebennik, and J. Pankratova, ‘Packing Irregular Polygons using Quasi Phi-functions’, in *2020 10th International Conference on Advanced Computer Information Technologies (ACIT)*, Deggendorf, Germany: IEEE, Sep. 2020, pp. 1–5. doi: 10.1109/ACIT49673.2020.9208979.

[38] J. Bennell, G. Scheithauer, Y. Stoyan, T. Romanova, and A. Pankratov, ‘Optimal clustering of a pair of irregular objects’, *J Glob Optim*, vol. 61, no. 3, pp. 497–524, Mar. 2015, doi: 10.1007/s10898-014-0192-0.

[39] A. Pankratov, T. Romanova, and I. Litvinchev, ‘Packing Oblique 3D Objects’, *Mathematics*, vol. 8, no. 7, p. 1130, Jul. 2020, doi: 10.3390/math8071130.

[40] Y. G. Stoyan, Gil ,N. I., Scheithauer ,G., Pankratov ,A., and I. and Magdalina, ‘Packing of convex polytopes into a parallelepiped’, *Optimization*, vol. 54, no. 2, pp. 215–235, Apr. 2005, doi: 10.1080/02331930500050681.

[41] Yu. Stoyan *et al.*, ‘Simulation of 3D Volume Filling with Non-Spherical and Spherical Titanium Alloy Powder Particles for Additive Manufacturing’, *Cybern Syst Anal*, vol. 60, no. 3, pp. 422–432, May 2024, doi: 10.1007/s10559-024-00683-6.

[42] J. Bennell, I. Litvinchev, A. Pankratov, and T. Romanova, ‘Packing stretched convex polygons in an optimized rectangle’, *Wireless Netw*, vol. 30, no. 9, pp. 7369–7376, Dec. 2024, doi: 10.1007/s11276-023-03642-9.

[43] Y. E. Stoian, A. M. Chugay, A. V. Pankratov, and T. E. Romanova, ‘Two Approaches to Modeling and Solving the Packing Problem for Convex Polytopes’, *Cybern Syst Anal*, vol. 54, no. 4, pp. 585–593, Jul. 2018, doi: 10.1007/s10559-018-0059-3.

[44] Y. G. Stoyan and A. M. Chugay, ‘Multistage Approach to Solving the Optimization Problem of Packing Nonconvex Polyhedra’, *Cybern Syst Anal*, vol. 56, no. 2, pp. 259–268, Mar. 2020, doi: 10.1007/s10559-020-00241-w.

[45] J. A. Bennell and J. F. Oliveira, ‘The geometry of nesting problems: A tutorial’, *European Journal of Operational Research*, vol. 184, no. 2, pp. 397–415, Jan. 2008, doi: 10.1016/j.ejor.2006.11.038.

[46] Y. Stoyan, A. Pankratov, and T. Romanova, ‘Quasi-phi-functions and optimal packing of ellipses’, *J Glob Optim*, vol. 65, no. 2, pp. 283–307, Jun. 2016, doi: 10.1007/s10898-015-0331-2.

[47] X. Jia and R. A. Williams, ‘A packing algorithm for particles of arbitrary shapes’, *Powder Technology*, vol. 120, no. 3, pp. 175–186, Oct. 2001, doi: 10.1016/S0032-5910(01)00268-6.

[48] T. Byholm, M. Toivakka, and J. Westerholm, ‘Effective packing of 3-dimensional voxel-based arbitrarily shaped particles’, *Powder Technology*, vol. 196, no. 2, pp. 139–146, Dec. 2009, doi: 10.1016/j.powtec.2009.07.013.

[49] J. Egeblad, C. Garavelli, S. Lisi, and D. Pisinger, ‘Heuristics for container loading of furniture’, *European Journal of Operational Research*, vol. 200, no. 3, pp. 881–892, Feb. 2010, doi: 10.1016/j.ejor.2009.01.048.

[50] G. Fasano, ‘A global optimization point of view to handle non-standard object packing problems’, *J Glob Optim*, vol. 55, no. 2, pp. 279–299, Feb. 2013, doi: 10.1007/s10898-012-9865-8.

[51] Y.-K. Joung and S. Do Noh, ‘Intelligent 3D packing using a grouping algorithm for automotive container engineering’, *Journal of Computational Design and Engineering*, vol. 1, no. 2, pp. 140–151, Apr. 2014, doi: 10.7315/JCDE.2014.014.

[52] X. Liu, J. Liu, A. Cao, and Z. Yao, ‘HAPE3D—a new constructive algorithm for the 3D irregular packing problem’, *Frontiers Inf Technol Electronic Eng*, vol. 16, no. 5, pp. 380–390, May 2015, doi: 10.1631/FITEE.1400421.

[53] M. Verkhoturov, A. Petunin, G. Verkhoturova, K. Danilov, and D. Kurennov, ‘The 3D Object Packing Problem into a Parallelepiped Container Based on Discrete-Logical Representation’, *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 1–5, Jan. 2016, doi: 10.1016/j.ifacol.2016.07.540.

[54] Y. Ma, Z. Chen, W. Hu, and W. Wang, ‘Packing Irregular Objects in 3D Space via Hybrid Optimization’, *Computer Graphics Forum*, vol. 37, no. 5, pp. 49–59, 2018, doi: 10.1111/cgf.13490.

[55] F. Wang and K. Hauser, ‘Stable bin packing of non-convex 3D objects with a robot manipulator’, Dec. 10, 2018, *arXiv*: arXiv:1812.04093. doi: 10.48550/arXiv.1812.04093.

[56] C. Carrick and I. Y. Kim, ‘Packaging optimization using the dynamic vector fields method’, *Numerical Meth Engineering*, vol. 120, no. 7, pp. 860–879, Nov. 2019, doi: 10.1002/nme.6161.

[57] V. Chekanin, ‘Solving the Problem of Packing Objects of Complex Geometric Shape into a Container of Arbitrary Dimension’, *Proceedings of the 30th International Conference on Computer Graphics and Machine Vision (GraphiCon 2020). Part 2*, pp. paper50-1-paper50-13, Dec. 2020, doi: 10.51130/graphicon-2020-2-3-50.

[58] Q. Cui, V. Rong, D. Chen, and W. Matusik, ‘Dense, Interlocking-Free and Scalable Spectral Packing of Generic 3D Objects’, *ACM Trans. Graph.*, vol. 42, no. 4, p. 141:1-141:14, Jul. 2023, doi: 10.1145/3592126.

[59] Q. Zhuang, Z. Chen, K. He, J. Cao, and W. Wang, ‘Dynamics simulation-based packing of irregular 3D objects’, *Computers & Graphics*, vol. 123, p. 103996, Oct. 2024, doi: 10.1016/j.cag.2024.103996.

[60] C. Douglas, J. S. Huh, S. O. Jun, and I. Y. Kim, ‘Packing optimization of practical systems using a dynamic acceleration methodology’, *Journal of Engineering and Applied Science*, vol. 71, no. 1, p. 92, Apr. 2024, doi: 10.1186/s44147-024-00426-6.

[61] S.-M. Hur, K.-H. Choi, S.-H. Lee, and P.-K. Chang, ‘Determination of fabricating orientation and packing in SLS process’, *Journal of Materials Processing Technology*, vol. 112, no. 2, pp. 236–243, May 2001, doi: 10.1016/S0924-0136(01)00581-7.

[62] A. S. Gogate and S. S. Pande, ‘Intelligent layout planning for rapid prototyping’, *International Journal of Production Research*, vol. 46, no. 20, pp. 5607–5631, Jan. 2008, doi: 10.1080/00207540701277002.

[63] S. Tiwari, G. Fadel, and P. Fenyes, ‘A Fast and Efficient Compact Packing Algorithm for SAE and ISO Luggage Packing Problems’, *Journal of Computing and Information Science in Engineering*, vol. 10, no. 2, p. 021010, Jun. 2010, doi: 10.1115/1.3330440.

[64] M. I. Campbell, C. H. Amon, and J. Cagan, ‘Optimal Three-Dimensional Placement of Heat Generating Electronic Components’, *Journal of Electronic Packaging*, vol. 119, no. 2, pp. 106–113, Jun. 1997, doi: 10.1115/1.2792210.

[65] S. Szykman and J. Cagan, ‘Constrained Three-Dimensional Component Layout Using Simulated Annealing’, *Journal of Mechanical Design*, vol. 119, no. 1, pp. 28–35, Mar. 1997, doi: 10.1115/1.2828785.

[66] J. Cagan, D. Degentesh, and S. Yin, ‘A simulated annealing-based algorithm using hierarchical models for general three-dimensional component layout’, *Computer-Aided Design*, vol. 30, no. 10, pp. 781–790, Sep. 1998, doi: 10.1016/S0010-4485(98)00036-0.

[67] X. Zhang, B. Zhou, Y. Zeng, and P. Gu, ‘Model layout optimization for solid ground curing rapid prototyping processes’, *Robotics and Computer-Integrated Manufacturing*, vol. 18, no. 1, pp. 41–51, Feb. 2002, doi: 10.1016/S0736-5845(01)00022-9.

[68] S. H. Jang and K. Y. Rhee, ‘An Application of Annealing Algorithm to the Layout Design of Ocean Space Submergible Boat’, *JSME International Journal*, vol. 47, no. 2, 2004.

[69] J. E. Lewis, R. K. Ragade, A. Kumar, and W. E. Biles, ‘A distributed chromosome genetic algorithm for bin-packing’, *Robotics and Computer-Integrated Manufacturing*, vol. 21, no. 4, pp. 486–495, Aug. 2005, doi: 10.1016/j.rcim.2004.11.017.

[70] C. Aladahalli, J. Cagan, and K. Shimada, ‘Objective Function Effect Based Pattern Search—An Implementation for 3D Component Layout’, *Journal of Mechanical Design*, vol. 129, no. 3, pp. 255–265, Dec. 2005, doi: 10.1115/1.2406096.

[71] C. Aladahalli, J. Cagan, and K. Shimada, ‘Objective Function Effect Based Pattern Search—Theoretical Framework Inspired by 3D Component Layout’, *Journal of Mechanical Design*, vol. 129, no. 3, pp. 243–254, Mar. 2006, doi: 10.1115/1.2406095.

[72] J. Egeblad, B. K. Nielsen, and A. Odgaard, ‘Fast neighborhood search for two- and three-dimensional nesting problems’, *European Journal of Operational Research*, vol. 183, no. 3, pp. 1249–1266, Dec. 2007, doi: 10.1016/j.ejor.2005.11.063.

[73] W. Yang, W. Liu, L. Liu, and A. Xu, ‘A Genetic Algorithm for Automatic Packing in Rapid Prototyping Processes’, in *Advanced Intelligent Computing Theories and Applications. With Aspects of Theoretical and Methodological Issues*, vol. 5226, D.-S. Huang, D. C. Wunsch, D. S. Levine, and K.-H. Jo, Eds, in Lecture Notes in Computer Science, vol. 5226. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1072–1077. doi: 10.1007/978-3-540-87442-3_132.

[74] J. Egeblad, ‘Placement of two- and three-dimensional irregular shapes for inertia moment and balance’, *International Transactions in Operational Research*, vol. 16, no. 6, pp. 789–807, 2009, doi: 10.1111/j.1475-3995.2009.00703.x.

[75] J. Egeblad, B. K. Nielsen, and M. Brazil, ‘Translational packing of arbitrary polytopes’, *Computational Geometry*, vol. 42, no. 4, pp. 269–288, May 2009, doi: 10.1016/j.comgeo.2008.06.003.

[76] P. Nebel, G. Richter, and K. Weicker, ‘Multi-container loading with non-convex 3D shapes using a GA/TS hybrid’, in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, in GECCO ’12. New York, NY, USA: Association for Computing Machinery, Jul. 2012, pp. 1143–1150. doi: 10.1145/2330163.2330321.

[77] S. Wu, M. Kay, R. King, A. Vila-parrish, and D. Warsing, 'Multi-objective optimization of 3D packing problem in additive manufacturing', *IIE Annual Conference and Expo 2014*, pp. 1485–1494, Jan. 2014.

[78] S. Edelkamp and P. Wichern, 'Packing Irregular-Shaped Objects for 3D Printing', in *KI 2015: Advances in Artificial Intelligence*, S. Hölldobler, R. Peñaloza, and S. Rudolph, Eds, Cham: Springer International Publishing, 2015, pp. 45–58. doi: 10.1007/978-3-319-24489-1_4.

[79] C. Zhao, L. Jiang, and K. L. Teo, 'A hybrid chaos firefly algorithm for three-dimensional irregular packing problem', *JIMO*, vol. 16, no. 1, pp. 409–429, Oct. 2018, doi: 10.3934/jimo.2018160.

[80] L. J. P. Araújo, A. Panesar, E. Özcan, J. Atkin, M. Baumers, and I. Ashcroft, 'An experimental analysis of deepest bottom-left-fill packing methods for additive manufacturing', *International Journal of Production Research*, vol. 58, no. 22, pp. 6917–6933, Nov. 2020, doi: 10.1080/00207543.2019.1686187.

[81] Y. Zhao, C. Rausch, and C. Haas, 'Optimizing 3D Irregular Object Packing from 3D Scans Using Metaheuristics', *Advanced Engineering Informatics*, vol. 47, p. 101234, Jan. 2021, doi: 10.1016/j.aei.2020.101234.

[82] J. Zhang, Yao ,Xifan, and Y. and Li, 'Improved evolutionary algorithm for parallel batch processing machine scheduling in additive manufacturing', *International Journal of Production Research*, vol. 58, no. 8, pp. 2263–2282, Apr. 2020, doi: 10.1080/00207543.2019.1617447.

[83] H. Kim, Y. Lee, S. Han, and J. Oh, *Hybrid Genetic Algorithm for Packing Segments of Decommissioned Nuclear Reactor Components*. 2022.

[84] Z. Lu, K. Hu, and T. S. Ng, 'Improving Additive Manufacturing production planning: A sub-second pixel-based packing algorithm', *Computers & Industrial Engineering*, vol. 181, p. 109318, Jul. 2023, doi: 10.1016/j.cie.2023.109318.

[85] H. Liu, L. Zhou, J. Yang, and J. Zhao, 'The 3D bin packing problem for multiple boxes and irregular items based on deep Q-network', *Appl Intell*, vol. 53, no. 20, pp. 23398–23425, Oct. 2023, doi: 10.1007/s10489-023-04604-6.

[86] A. Petrowski and S. Ben-Hamida, *Evolutionary Algorithms*. John Wiley & Sons, 2017.

[87] M. Gendreau and J.-Y. Potvin, Eds, *Handbook of Metaheuristics*, vol. 272. in International Series in Operations Research & Management Science, vol. 272. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-319-91086-4.

[88] R. Hooke and T. A. Jeeves, '` ` Direct Search'' Solution of Numerical and Statistical Problems', *J. ACM*, vol. 8, no. 2, pp. 212–229, Apr. 1961, doi: 10.1145/321062.321069.

[89] S. Tao, C. Wu, Z. Sheng, and X. Wang, 'Stochastic Project Scheduling with Hierarchical Alternatives', *Applied Mathematical Modelling*, vol. 58, pp. 181–202, Jun. 2018, doi: 10.1016/j.apm.2017.09.015.

[90] K. Rajwar, K. Deep, and S. Das, 'An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges',

Artif Intell Rev, vol. 56, no. 11, pp. 13187–13257, Nov. 2023, doi: 10.1007/s10462-023-10470-y.

[91] B. Toaza and D. Esztergár-Kiss, 'A review of metaheuristic algorithms for solving TSP-based scheduling optimization problems', *Applied Soft Computing*, vol. 148, p. 110908, Nov. 2023, doi: 10.1016/j.asoc.2023.110908.

[92] Thingiverse.com, 'Low Poly Frog by TK3DPrinting', Thingiverse. Accessed: Apr. 16, 2025. [Online]. Available: https://www.thingiverse.com/thing:1762864

[93] H. T. Dean, Y. Tu, and J. F. Raffensperger, 'An improved method for calculating the no-fit polygon', *Computers & Operations Research*, vol. 33, no. 6, pp. 1521–1539, Jun. 2006, doi: 10.1016/j.cor.2004.11.005.

[94] E. K. Burke, R. S. R. Hellier, G. Kendall, and G. Whitwell, 'Complete and robust no-fit polygon generation for the irregular stock cutting problem', *European Journal of Operational Research*, vol. 179, no. 1, pp. 27–49, May 2007, doi: 10.1016/j.ejor.2006.03.011.

[95] Q. Luo and Y. Rao, 'Improved Sliding Algorithm for Generating No-Fit Polygon in the 2D Irregular Packing Problem', *Mathematics*, vol. 10, no. 16, Art. no. 16, Jan. 2022, doi: 10.3390/math10162941.

[96] M. Aleksandrov, S. Zlatanova, and D. J. Heslop, 'Voxelisation Algorithms and Data Structures: A Review', *Sensors*, vol. 21, no. 24, Art. no. 24, Jan. 2021, doi: 10.3390/s21248241.

[97] J. Wang, B. Zhang, S. Yang, and Y. Chen, 'Dynamic mode decomposition coupled with multilevel octree grid algorithm for large-scale discrete ordinates neutron transport calculations', *Progress in Nuclear Energy*, vol. 172, p. 105210, Jul. 2024, doi: 10.1016/j.pnucene.2024.105210.

[98] Y. Zhao, 'Optimal Packing of Irregular 3D Objects For Transportation and Disposal', 2022.

[99] R. A. Williams, X. Jia, P. Ikin, and D. Knight, 'Use of multiscale particle simulations in the design of nuclear plant decommissioning', *Particuology*, vol. 9, no. 4, pp. 358–364, Aug. 2011, doi: 10.1016/j.partic.2010.10.003.

[100] Thingiverse.com, 'Floating Head Heat Exchanger by Scott Hollister aka sdholli by Tse_Tso', Thingiverse. Accessed: Apr. 01, 2025. [Online]. Available: https://www.thingiverse.com/thing:4836720

[101] A. Bortfeldt and G. Wäscher, 'Constraints in container loading – A state-of-the-art review', *European Journal of Operational Research*, vol. 229, no. 1, pp. 1–20, Aug. 2013, doi: 10.1016/j.ejor.2012.12.006.

[102] 'WPS/300/05 Specification for Waste Packages Containing Low Heat Generating Waste Part D - Container Specific Requirements', GOV.UK. Accessed: Mar. 28, 2025. [Online]. Available: https://www.gov.uk/government/publications/specification-for-waste-packages-containing-low-heat-generating-waste-part-d-container-specific-requirements

[103] Z. Xu, M. Li, B. Zou, and M. Yang, 'A GPU-based point kernel gamma dose rate computing code for virtual simulation in radiation-controlled area', *Nuclear Engineering and Technology*, vol. 55, no. 6, pp. 1966–1973, Jun. 2023, doi: 10.1016/j.net.2023.03.018.

[104] P. Sun, H. Zhang, L. Xing, B. Zhong, Y. Ying, and H. Shen, 'NPTS-PK: A new point kernel code for fast calculation of 3D gamma radiation field', *Radiation Physics and Chemistry*, vol. 226, p. 112329, Jan. 2025, doi: 10.1016/j.radphyschem.2024.112329.

[105] 'Methodology to Manage Material and Waste from Nuclear Decommissioning - World Nuclear Association'. Accessed: Jul. 07, 2025. [Online]. Available: https://world-nuclear.org/our-association/publications/working-group-reports/Methodology-to-Manage-Material-and-Waste-from-Nucl

[106] A. Shamir, 'A survey on Mesh Segmentation Techniques', *Computer Graphics Forum*, vol. 27, no. 6, pp. 1539–1556, Sep. 2008, doi: 10.1111/j.1467-8659.2007.01103.x.

[107] C. Araújo, D. Cabiddu, M. Attene, M. Livesu, N. Vining, and A. Sheffer, 'Surface2Volume: surface segmentation conforming assemblable volumetric partition', *ACM Trans. Graph.*, vol. 38, no. 4, pp. 1–16, Aug. 2019, doi: 10.1145/3306346.3323004.

[108] J. Hao, L. Fang, and R. Williams, 'An efficient curvature-based partitioning of large-scale STL models', *Rapid Prototyping Journal*, vol. 17, pp. 116–127, Mar. 2011, doi: 10.1108/13552541111113862.

[109] E. A. Yu, J. Yeom, C. C. Tutum, E. Vouga, and R. Miikkulainen, 'Evolutionary decomposition for 3D printing', in *Proceedings of the Genetic and Evolutionary Computation Conference*, in GECCO '17. New York, NY, USA: Association for Computing Machinery, Jul. 2017, pp. 1272–1279. doi: 10.1145/3071178.3071310.

[110] 'Backgrounder on Decommissioning Nuclear Power Plants', NRC Web. Accessed: Jul. 02, 2025. [Online]. Available: https://www.nrc.gov/reading-rm/doc-collections/fact-sheets/decommissioning.html

[111] Y. Oh, C. Zhou, and S. Behdad, 'Part decomposition and assembly-based (Re) design for additive manufacturing: A review', *Additive Manufacturing*, vol. 22, pp. 230–242, Aug. 2018, doi: 10.1016/j.addma.2018.04.018.

[112] I. Kim *et al.*, 'A framework for a flexible cutting-process simulation of a nuclear facility decommission', *Annals of Nuclear Energy*, vol. 97, pp. 204–207, Nov. 2016, doi: 10.1016/j.anucene.2016.07.004.

[113] D. Hyun *et al.*, 'A methodology to simulate the cutting process for a nuclear dismantling simulation based on a digital manufacturing platform', *Annals of Nuclear Energy*, vol. 103, pp. 369–383, May 2017, doi: 10.1016/j.anucene.2017.01.035.

[114] 'Radiological protection of people and the environment: generic developed principles', GOV.UK. Accessed: Jan. 19, 2026. [Online]. Available: https://www.gov.uk/government/publications/rsr-generic-developed-principles-regulatory-assessment/radiological-protection-of-people-and-the-environment-generic-developed-principles

[115] *Nuclear Decommissioning, Waste Management, and Environmental Site Remediation*. Elsevier, 2003. doi: 10.1016/B978-0-7506-7744-8.X5000-0.

[116] Internationale Atomenergie-Organisation, Ed., *Radiological characterization of shut down nuclear reactors for decommissioning purposes*. in Technical reports series / International Atomic Energy Agency, no. 389. Vienna: International Atomic Energy Agency, 1998.

[117] ‘Radiological Worker Training’. Accessed: Jul. 03, 2025. [Online]. Available: https://www.standards.doe.gov/standards-documents/1100/1130-bhdbk-2007

[118] N. Chao, Y. Liu, H. Xia, A. Ayodeji, and H. Yang, ‘Adaptive point kernel dose assessment method for cutting simulation on irregular geometries in nuclear facility decommissioning’, *Radiation Physics and Chemistry*, vol. 150, pp. 125–136, Sep. 2018, doi: 10.1016/j.radphyschem.2018.04.035.

[119] L. Yang, Y. Liu, M. Peng, A. Ayodeji, and N. Chao, ‘Voxel-based point kernel method for dose rate assessment of non-uniform activity and self-shielding sources in nuclear facility decommissioning’, *Radiation Physics and Chemistry*, vol. 164, p. 108381, Nov. 2019, doi: 10.1016/j.radphyschem.2019.108381.

[120] ‘Barsebäck reactor vessel dismantled’, World Nuclear News. Accessed: Jul. 04, 2025. [Online]. Available: https://world-nuclear-news.org/articles/barseback-reactor-vessel-dismantled

[121] I. A. E. Agency, ‘Dismantling of Contaminated Stacks at Nuclear Facilities’, International Atomic Energy Agency, Text, 2005. Accessed: Jul. 04, 2025. [Online]. Available: https://www.iaea.org/publications/7238/dismantling-of-contaminated-stacks-at-nuclear-facilities

[122] H. Moon, S. Mirmotalebi, Y. Jang, Y. Ahn, and N. Kwon, ‘Risk Evaluation of Radioactive Concrete Structure Decommissioning in Nuclear Power Plants Using Fuzzy-AHP’, *Buildings*, vol. 14, no. 6, Art. no. 6, Jun. 2024, doi: 10.3390/buildings14061536.

[123] C. Halliwell and 1628 E. Southern Avenue WM Symposia, ‘The Windscale Advanced Gas Cooled Reactor (WAGR) Decommissioning Project A Close Out Report for WAGR Decommissioning Campaigns 1 to 10 - 12474’, WM Symposia, 1628 E. Southern Avenue, Suite 9-332, Tempe, AZ 85282 (United States), INIS-US--14-WM-12474, 2012. Accessed: Jul. 04, 2025. [Online]. Available: https://inis.iaea.org/records/epe0b-8bb70

[124] ‘Nuclear Decommissioning at Sellafield using LaserSnake’. Accessed: Jul. 04, 2025. [Online]. Available: https://www.twi-global.com/media-and-events/insights/laser-cutting-for-nuclear-decommissioning-at-sellafield

[125] ‘Revolutionising nuclear decommissioning: a world-first in robotic size reduction? - RAICo’. Accessed: Jul. 04, 2025. [Online]. Available: https://raico.org/revolutionising-nuclear-decommissioning-a-world-first-in-robotic-size-reduction/

[126] ‘Robotic size reduction of waste with laser cutting’, AtkinsRéalis. Accessed: Jul. 04, 2025. [Online]. Available: https://www.atkinsrealis.com/en/projects/robotic-size-reduction-with-laser-cutting

[127] ‘NuPlant - New Build and Decommissioning Planning Innovation’. Accessed: Mar. 28, 2025. [Online]. Available: https://www.structurevision.com/radioactive_waste_disposal.htm

[128] C. Xu *et al.*, ‘Property Predictions for Packed Columns Using Monte Carlo and Discrete Element Digital Packing Algorithms’, *CMES*, vol. 23, no. 2, pp. 117–126, 2008, doi: 10.3970/cmes.2008.023.117.

[129] B. Naylor, ‘A Tutorial on Binary Space Partitioning Trees’, Jan. 2005.

[130] ‘File:Binary space partition.png - Wikimedia Commons’. Accessed: Jun. 18, 2025. [Online]. Available: https://commons.wikimedia.org/wiki/File:Binary_space_partition.png

[131] E. A. Yu, J. Yeom, C. C. Tutum, E. Vouga, and R. Miikkulainen, ‘Evolutionary decomposition for 3D printing’, in *Proceedings of the Genetic and Evolutionary Computation Conference*, in GECCO ’17. New York, NY, USA: Association for Computing Machinery, Jul. 2017, pp. 1272–1279. doi: 10.1145/3071178.3071310.

[132] L. Luo, I. Baran, S. Rusinkiewicz, and W. Matusik, ‘Chopper: partitioning models into 3D-printable parts’, *ACM Trans. Graph.*, vol. 31, no. 6, p. 129:1-129:9, Nov. 2012, doi: 10.1145/2366145.2366148.

[133] T. A. de Queiroz, F. K. Miyazawa, Y. Wakabayashi, and E. C. Xavier, ‘Algorithms for 3D guillotine cutting problems: Unbounded knapsack, cutting stock and strip packing’, *Computers & Operations Research*, vol. 39, no. 2, pp. 200–212, Feb. 2012, doi: 10.1016/j.cor.2011.03.011.

[134] W. Han, J. A. Bennell, X. Zhao, and X. Song, ‘Construction heuristics for two-dimensional irregular shape bin packing with guillotine constraints’, *European Journal of Operational Research*, vol. 230, no. 3, pp. 495–504, Nov. 2013, doi: 10.1016/j.ejor.2013.04.048.

[135] X. Jia, ‘Simulating the Dismantling/Packing Process using NuPlant’, University of Leeds, 2009.

[136] ‘DigiPac - Structure Vision’. Accessed: Jun. 18, 2025. [Online]. Available: https://www.structurevision.com/digipac.htm

[137] S. Tiwari, G. Fadel, and P. Fenyes, ‘A Fast and Efficient Compact Packing Algorithm for SAE and ISO Luggage Packing Problems’, *Journal of Computing and Information Science in Engineering*, vol. 10, no. 2, p. 021010, Jun. 2010, doi: 10.1115/1.3330440.

[138] T. Byholm, M. Toivakka, and J. Westerholm, ‘Effective packing of 3-dimensional voxel-based arbitrarily shaped particles’, *Powder Technology*, vol. 196, no. 2, pp. 139–146, Dec. 2009, doi: 10.1016/j.powtec.2009.07.013.

[139] F. Wang and K. Hauser, ‘Stable bin packing of non-convex 3D objects with a robot manipulator’, Dec. 10, 2018, *arXiv*: arXiv:1812.04093. doi: 10.48550/arXiv.1812.04093.

[140] Nojhan, *Français : Optimum de Pareto*. 2006. Accessed: Jul. 17, 2025. [Online]. Available: https://commons.wikimedia.org/wiki/File:Front_pareto.svg

[141] S. Katoch, S. S. Chauhan, and V. Kumar, ‘A review on genetic algorithm: past, present, and future’, *Multimed Tools Appl*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021, doi: 10.1007/s11042-020-10139-6.

[142] A. Lambora, K. Gupta, and K. Chopra, ‘Genetic Algorithm- A Literature Review’, in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, Feb. 2019, pp. 380–384. doi: 10.1109/COMITCon.2019.8862255.

[143] M. Albadr, S. Tiun, M. Ayob, and F. Al-Dhief, ‘Genetic Algorithm Based on Natural Selection Theory for Optimization Problems’, *Symmetry*, vol. 12, pp. 1–31, Oct. 2020, doi: 10.3390/sym12111758.

[144] B. Marinus, ‘Multidisciplinary Optimization of Aircraft Propeller Blades’, 2011.

[145] N. Razali and J. Geraghty, *Genetic Algorithm Performance with Different Selection Strategies in Solving TSP*, vol. 2. 2011.

[146] M. Namdari, H. Jazayeri-Rad, and S.-J. Hashemi, 'Process Fault Diagnosis Using Support Vector Machines with a Genetic Algorithm based Parameter Tuning', *Journal of Automation and Control*, vol. 2, no. 1, Art. no. 1, Jan. 2014, doi: 10.12691/automation-2-1-1.

[147] K. Jebari, 'Parent Selection Operators for Genetic Algorithms', *International Journal of Engineering Research & Technology*, vol. 12, pp. 1141–1145, Nov. 2013.

[148] manlio, 'Answer to "What is the difference between roulette wheel selection, rank selection and tournament selection?"', Stack Overflow. Accessed: Jul. 21, 2025. [Online]. Available: https://stackoverflow.com/a/23193432

[149] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. in Natural Computing Series. Berlin, Heidelberg: Springer, 2015. doi: 10.1007/978-3-662-44874-8.

[150] B. Alhijawi and A. Awajan, 'Genetic algorithms: theory, genetic operators, solutions, and applications', *Evol. Intel.*, vol. 17, no. 3, pp. 1245–1256, Jun. 2024, doi: 10.1007/s12065-023-00822-6.

[151] 'Genetic Algorithms Crossover Techniques'. Accessed: Jul. 21, 2025. [Online]. Available: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm

[152] P. Kora and P. Yadlapalli, 'Crossover Operators in Genetic Algorithms: A Review', *International Journal of Computer Applications*, vol. 162, pp. 34–36, Mar. 2017, doi: 10.5120/ijca2017913370.

[153] L. Eshelman, R. Caruana, and J. Schaffer, *Biases in the Crossover Landscape*. 1989, p. 19.

[154] W. M. Spears and K. A. De Jong, 'An Analysis of Multi-Point Crossover', in *Foundations of Genetic Algorithms*, vol. 1, G. J. E. Rawlins, Ed., Elsevier, 1991, pp. 301–315. doi: 10.1016/B978-0-08-050684-5.50022-7.

[155] S. O'Hagan, J. Knowles, and D. B. Kell, 'Exploiting genomic knowledge in optimising molecular breeding programmes: algorithms from evolutionary computing', *PLoS One*, vol. 7, no. 11, p. e48862, 2012, doi: 10.1371/journal.pone.0048862.

[156] K. L. Johnson and N. B. Carnegie, 'Calibration of an Adaptive Genetic Algorithm for Modeling Opinion Diffusion', *Algorithms*, vol. 15, no. 2, Art. no. 2, Feb. 2022, doi: 10.3390/a15020045.

[157] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, 'A fast and elitist multiobjective genetic algorithm: NSGA-II', *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002, doi: 10.1109/4235.996017.

[158] A. Guerreiro, C. Fonseca, and L. Paquete, 'The Hypervolume Indicator: Computational Problems and Algorithms', *ACM Computing Surveys*, vol. 54, pp. 1–42, Jul. 2021, doi: 10.1145/3453474.

[159] E. Zitzler and L. Thiele, 'Thiele, L.: Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. IEEE Trans. on Evolutionary

Computation 3, 257-271', *IEEE Transactions on Evolutionary Computation*, vol. 3, Oct. 2000, doi: 10.1109/4235.797969.

[160] E. Zitzler and L. Thiele, 'Multiobjective optimization using evolutionary algorithms — A comparative case study', in *Parallel Problem Solving from Nature — PPSN V*, A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, Eds, Berlin, Heidelberg: Springer, 1998, pp. 292–301. doi: 10.1007/BFb0056872.

[161] 'Computation of the Hypervolume Indicator'. Accessed: May 13, 2025. [Online]. Available: https://lopez-ibanez.eu/hypervolume

[162] K. Deb, 'An efficient constraint handling method for genetic algorithms', *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2, pp. 311–338, Jun. 2000, doi: 10.1016/S0045-7825(99)00389-8.

[163] A. Ponsich, C. Azzaro-Pantel, S. Domenech, and L. Pibouleau, 'Constraint handling strategies in Genetic Algorithms application to optimal batch plant design', *Chemical Engineering and Processing: Process Intensification*, vol. 47, no. 3, pp. 420–434, Mar. 2008, doi: 10.1016/j.cep.2007.01.020.

[164] C. A. Coello Coello, 'Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art', *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11, pp. 1245–1287, Jan. 2002, doi: 10.1016/S0045-7825(01)00323-1.

[165] H. Terashima-Marín, P. Ross, C. J. Farías-Zárate, E. López-Camacho, and M. Valenzuela-Rendón, 'Generalized hyper-heuristics for solving 2D Regular and Irregular Packing Problems', *Ann Oper Res*, vol. 179, no. 1, pp. 369–392, Sep. 2010, doi: 10.1007/s10479-008-0475-2.

[166] J. C. Gomez and H. Terashima-Marín, 'Approximating Multi-Objective Hyper-Heuristics for Solving 2D Irregular Cutting Stock Problems', in *Advances in Soft Computing*, G. Sidorov, A. Hernández Aguirre, and C. A. Reyes García, Eds, Berlin, Heidelberg: Springer, 2010, pp. 349–360. doi: 10.1007/978-3-642-16773-7_30.

[167] E. López-Camacho, H. Terashima-Marín, and P. Ross, 'Defining a Problem-State Representation with Data Mining within a Hyper-heuristic Model Which Solves 2D Irregular Bin Packing Problems', in *Advances in Artificial Intelligence – IBERAMIA 2010*, A. Kuri-Morales and G. R. Simari, Eds, Berlin, Heidelberg: Springer, 2010, pp. 204–213. doi: 10.1007/978-3-642-16952-6_21.

[168] E. López-Camacho, H. Terashima-Marín, and P. Ross, 'A hyper-heuristic for solving one and two-dimensional bin packing problems', in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, in GECCO '11. New York, NY, USA: Association for Computing Machinery, Jul. 2011, pp. 257–258. doi: 10.1145/2001858.2002003.

[169] E. López-Camacho, H. Terashima-Marin, P. Ross, and G. Ochoa, 'A unified hyper-heuristic framework for solving bin packing problems', *Expert Systems with Applications*, vol. 41, no. 15, pp. 6876–6889, Nov. 2014, doi: 10.1016/j.eswa.2014.04.043.

[170] J. C. Gomez and H. Terashima-Marín, 'Evolutionary hyper-heuristics for tackling bi-objective 2D bin packing problems', *Genet Program Evolvable Mach*, vol. 19, no. 1, pp. 151–181, Jun. 2018, doi: 10.1007/s10710-017-9301-4.

[171] F. Guerriero and F. P. Saccomanno, ‘A hierarchical hyper-heuristic for the bin packing problem’, *Soft Comput*, vol. 27, no. 18, pp. 12997–13010, Sep. 2023, doi: 10.1007/s00500-022-07118-4.

[172] D. Domović, T. Rolich, and M. Golub, ‘Evolutionary hyper-heuristic for solving the strip-packing problem’, *The Journal of The Textile Institute*, vol. 110, no. 8, pp. 1141–1151, Aug. 2019, doi: 10.1080/00405000.2018.1550136.

[173] J. Williams, ‘Packing Optimisation with Hyper-Heuristics.’, Master’s Thesis, University of Leeds, 2023.

[174] ‘Data sets – ESICUP – EURO Special Interest Group on Cutting and Packing’. Accessed: May 15, 2025. [Online]. Available: https://www.euro-online.org/websites/esicup/data-sets/

[175] A. Moraglio and R. Poli, ‘Geometric crossover for the permutation representation’, *Intelligenza Artificiale*, vol. 5, no. 1, pp. 49–63, Feb. 2011, doi: 10.3233/IA-2011-0004.

[176] A. Martinez-Sykora, R. Alvarez-Valdes, J. A. Bennell, R. Ruiz, and J. M. Tamarit, ‘Matheuristics for the irregular bin packing problem with free rotations’, *European Journal of Operational Research*, vol. 258, no. 2, pp. 440–455, Apr. 2017, doi: 10.1016/j.ejor.2016.09.043.

[177] B. Guo, Z. Liang, Q. Peng, Y. Li, and F. Wu, ‘Irregular Packing Based on Principal Component Analysis Methodology’, *IEEE Access*, vol. 6, pp. 62675–62686, 2018, doi: 10.1109/ACCESS.2018.2876546.

[178] K. Fleszar and K. S. Hindi, ‘New heuristics for one-dimensional bin-packing’, *Computers & Operations Research*, vol. 29, no. 7, pp. 821–839, Jun. 2002, doi: 10.1016/S0305-0548(00)00082-4.

[179] A. Caprara and U. Pferschy, ‘Worst-case analysis of the subset sum algorithm for bin packing’, *Operations Research Letters*, vol. 32, no. 2, pp. 159–166, Mar. 2004, doi: 10.1016/S0167-6377(03)00092-0.

[180] A. Knight, ‘Optimised Disassembly Sequencing for 2D Cutting and Packing’, Master’s Thesis, University of Leeds, 2025.

[181] A. ElSayed, E. Kongar, S. Gupta, and T. Sobh, *An Online Genetic Algorithm for Automated Disassembly Sequence Generation*, vol. 3. 2011. doi: 10.1115/DETC2011-48635.

[182] W. Xu, Q. Tang, J. Liu, Z. Liu, Z. Zhou, and D. Pham, ‘Disassembly sequence planning using discrete Bees algorithm for human-robot collaboration in remanufacturing’, *Robotics and Computer-Integrated Manufacturing*, vol. 62, p. 101860, Apr. 2020, doi: 10.1016/j.rcim.2019.101860.

[183] J. L. Rickli and J. A. Camelio, ‘Multi-objective partial disassembly optimization based on sequence feasibility’, *Journal of Manufacturing Systems*, vol. 32, no. 1, pp. 281–293, Jan. 2013, doi: 10.1016/j.jmsy.2012.11.005.

[184] H. Wan and V. Krishna Gonnuru, ‘Disassembly planning and sequencing for end-of-life products with RFID enriched information’, *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 3, pp. 112–118, Jun. 2013, doi: 10.1016/j.rcim.2012.05.001.

[185] K. Wang, X. Li, L. Gao, and A. Garg, ‘Partial disassembly line balancing for energy consumption and profit under uncertainty’, *Robotics and Computer-Integrated Manufacturing*, vol. 59, pp. 235–251, Oct. 2019, doi: 10.1016/j.rcim.2019.04.014.

[186] H. Güçdemir and M. A. Ilgin, ‘A part grouping-based approach for disassembly sequencing’, *Journal of Engineering Research*, vol. 11, no. 1, p. 100026, Mar. 2023, doi: 10.1016/j.jer.2023.100026.

[187] G. Q. Jin, W. D. Li, and K. Xia, ‘Disassembly Matrix for Liquid Crystal Displays Televisions’, *Procedia CIRP*, vol. 11, pp. 357–362, Jan. 2013, doi: 10.1016/j.procir.2013.07.015.

[188] W. D. Li, K. Xia, L. Gao, and K.-M. Chao, ‘Selective disassembly planning for waste electrical and electronic equipment with case studies on liquid crystaldisplays’, *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 4, pp. 248–260, Aug. 2013, doi: 10.1016/j.rcim.2013.01.006.

[189] K. Xia, L. Gao, L. Wang, W. Li, X. Li, and W. Ijomah, ‘Service-oriented disassembly sequence planning for electrical and electronic equipment waste’, *Electronic Commerce Research and Applications*, vol. 20, pp. 59–68, Nov. 2016, doi: 10.1016/j.elerap.2016.09.007.

[190] T. F. Go, D. A. Wahab, M. N. A. Rahman, and R. Ramli, ‘A Design Framework for End-of-Life Vehicles Recovery: Optimization of Disassembly Sequence Using Genetic Algorithms’, *AJES*, vol. 6, no. 4, pp. 350–356, Aug. 2010, doi: 10.3844/ajessp.2010.350.356.

[191] T. F. Go, D. A. Wahab, M. N. Ab. Rahman, R. Ramli, and A. Hussain, ‘Genetically optimised disassembly sequence for automotive component reuse’, *Expert Systems with Applications*, vol. 39, no. 5, pp. 5409–5417, Apr. 2012, doi: 10.1016/j.eswa.2011.11.044.

[192] S. Smith, G. Smith, and W.-H. Chen, ‘Disassembly sequence structure graphs: An optimal approach for multiple-target selective disassembly sequence planning’, *Advanced Engineering Informatics*, vol. 26, no. 2, pp. 306–316, Apr. 2012, doi: 10.1016/j.aei.2011.11.003.

[193] Y. Luo, Q. Peng, and P. Gu, ‘Integrated multi-layer representation and ant colony search for product selective disassembly planning’, *Computers in Industry*, vol. 75, pp. 13–26, Jan. 2016, doi: 10.1016/j.compind.2015.10.011.

[194] G. Jun, J. Zhong, Y. Li, B. Du, and S. Guo, ‘A hybrid artificial fish swam algorithm for disassembly sequence planning considering setup time’, *Assembly Automation*, vol. 39, Oct. 2018, doi: 10.1108/AA-12-2017-180.

[195] M. Chand and C. Ravi, ‘A state-of-the-art literature survey on artificial intelligence techniques for disassembly sequence planning’, *CIRP Journal of Manufacturing Science and Technology*, vol. 41, pp. 292–310, Apr. 2023, doi: 10.1016/j.cirpj.2022.11.017.

[196] L. He *et al.*, ‘Disassembly sequence planning of equipment decommissioning for industry 5.0: Prospects and Retrospects’, *Advanced Engineering Informatics*, vol. 62, p. 102939, Oct. 2024, doi: 10.1016/j.aei.2024.102939.

[197] J. Akl, S. Pericherla, and B. Calli, ‘Cut Sequencing Algorithm for Safely Disassembling Large Structures’, in *2023 62nd IEEE Conference on Decision and Control (CDC)*, Dec. 2023, pp. 8581–8588. doi: 10.1109/CDC49753.2023.10384301.

[198] A. Cebulla, T. Asfour, and T. Kröger, ‘Beyond Feasibility: Efficiently Planning Robotic Assembly Sequences That Minimize Assembly Path Lengths’, in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2024, pp. 4076–4083. doi: 10.1109/IROS58592.2024.10801475.

[199] Y. Tian *et al.*, ‘ASAP: Automated Sequence Planning for Complex Robotic Assembly with Physical Feasibility’, Feb. 29, 2024, *arXiv*: arXiv:2309.16909. doi: 10.48550/arXiv.2309.16909.

[200] R. B. McGhee and A. A. Frank, ‘On the stability properties of quadruped creeping gaits’, *Mathematical Biosciences*, vol. 3, pp. 331–351, Aug. 1968, doi: 10.1016/0025-5564(68)90090-4.

[201] A. Thakur, ‘Gaits in Legged Robots’, Mechanical Engineering Department, Indian Institute of Technology, Patna, 2017. [Online]. Available: https://www.iitp.ac.in/~athakur/courses/ME512/L11--Legged%20Robot%20Gaits.pdf

[202] J. Pedro Pedroso and M. Kubo, ‘Stochastic Tree Search: An Illustration with the Knapsack Problem’, Universidade do Porto, Technical Report, 2009. [Online]. Available: https://www.dcc.fc.up.pt/Pubs/TR09/dcc-2009-02.pdf

[203] Thingiverse.com, ‘VVER-1000 pressure vessel - Best Soviet reactor ever made by Ajrocket’, Thingiverse. Accessed: Jun. 11, 2025. [Online]. Available: https://www.thingiverse.com/thing:5323610

[204] L. D. Whitley, ‘The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best’, in *Proceedings of the 3rd International Conference on Genetic Algorithms*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Jun. 1989, pp. 116–123.

[205] T. Blickle and L. Thiele, ‘A comparison of selection schemes used in evolutionary algorithms’, *Evol. Comput.*, vol. 4, no. 4, pp. 361–394, Dec. 1996, doi: 10.1162/evco.1996.4.4.361.

[206] Thingiverse.com, ‘Glove Box Array for Medical or Manufacturing by joshgoreworks’, Thingiverse. Accessed: Apr. 01, 2025. [Online]. Available: https://www.thingiverse.com/thing:2072747

[207] ‘How do we manage radioactive waste?’, UK Radioactive Waste & Materials Inventory. Accessed: Mar. 28, 2025. [Online]. Available: https://ukinventory.nda.gov.uk/information-hub/about-radioactive-waste/how-do-we-manage-radioactive-waste/

[208] ‘Sci-Lab SG Glovebox System Series’. Vigor Gloveboxes. Accessed: Apr. 02, 2025. [Online]. Available: https://lambdasystem.pl/pliki do pobrania/ produkty/SciLab.pdf

[209] ‘Westinghouse Technology Systems Manual, Section 11.1, Steam Generator Water Level Control System’. United States Nuclear Regulatory Commision, 2011. Accessed: Apr. 02, 2025. [Online]. Available: https://www.nrc.gov/docs/ML1122/ML11223A293.pdf

[210] R. W. Henderson and L. Eickelpasch, ‘Cutting Reactor Pressure Vessels and their Internals - Trends on Selected Technologies - 10247’, 2010. Accessed: Apr. 02, 2025. [Online]. Available: https://www.semanticscholar.org/paper/Cutting-Reactor-Pressure-Vessels-and-their-Trends-Henderson-Eickelpasch/ce3c81ba06356262f9e120e36d390a9fcf8308a0

[211] ‘[PDF] Evaluation Methodology of Remote Dismantling Equipment for Reactor Pressure Vessel in Decommissioning Project | Semantic Scholar’. Accessed: Apr. 02, 2025. [Online]. Available: https://www.semanticscholar.org/paper/Evaluation-Methodology-of-Remote-Dismantling-for-in-Hyun-Choi/184e0977dff78411ecf4e494e512a92d903fd458

[212] Diatech Inc., *Heat exchanger Dry cutting by Diamond wire saw*, (Aug. 07, 2016). Accessed: Jun. 21, 2025. [Online Video]. Available: https://www.youtube.com/watch?v=YbxMTy7O41s

[213] K. Krishnakumar, ‘Micro-Genetic Algorithms For Stationary And Non-Stationary Function Optimization’, in *Intelligent Control and Adaptive Systems*, SPIE, Feb. 1990, pp. 289–296. doi: 10.1117/12.969927.

[214] C. A. Coello Coello Coello and G. Toscano Pulido, ‘A Micro-Genetic Algorithm for Multiobjective Optimization’, in *Evolutionary Multi-Criterion Optimization*, E. Zitzler, L. Thiele, K. Deb, C. A. Coello Coello, and D. Corne, Eds, Berlin, Heidelberg: Springer, 2001, pp. 126–140. doi: 10.1007/3-540-44719-9_9.

[215] S. Park and S. K. Park, ‘A micro-genetic algorithm (GA v1.7.1a) for combinatorial optimization of physics parameterizations in the Weather Research and Forecasting model (v4.0.3) for quantitative precipitation forecast in Korea’, *Geoscientific Model Development*, vol. 14, no. 10, pp. 6241–6255, Oct. 2021, doi: 10.5194/gmd-14-6241-2021.

[216] S. Kazarlis, ‘Combinatorial Hill Climbing Using Micro-Genetic Algorithms’, 2007, pp. 411–416. doi: 10.1007/978-1-4020-6264-3_71.

[217] G. Dozier, J. Bowen, and D. Bahler, ‘Solving small and large scale constraint satisfaction problems using a heuristic-based microgenetic algorithm’, in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, Jun. 1994, pp. 306–311 vol.1. doi: 10.1109/ICEC.1994.349934.

[218] G. Alvarez Ccopa, ‘Can we make genetic algorithms work in high-dimensionality problems’, Jan. 2002.

[219] M. G. Rojas, A. C. Olivera, J. A. Carballido, and P. J. Vidal, ‘Memetic micro-genetic algorithms for cancer data classification’, *Intelligent Systems with Applications*, vol. 17, p. 200173, Feb. 2023, doi: 10.1016/j.iswa.2022.200173.

[220] A. Eiben, R. Hinterding, and Z. Michalewicz, ‘Parameter control in evolutionary algorithms’, *IEEE Trans. Evolutionary Computation*, vol. 3, pp. 124–141, Aug. 1999, doi: 10.1109/4235.771166.

[221] K. Deb and S. Agrawal, ‘Understanding Interactions Among Genetic Algorithm Parameters’, *Foundations of Genetic Algorithms*, vol. 5, Aug. 2002.

[222] T. Bäck, ‘Optimal Mutation Rates in Genetic Search’, in *Proceedings of the 5th International Conference on Genetic Algorithms*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 2–8.

[223] R. Kumar, M. Memoria, A. Gupta, and M. Awasthi, *Critical Analysis of Genetic Algorithm under Crossover and Mutation Rate*. 2021, p. 980. doi: 10.1109/ICAC3N53548.2021.9725640.

[224] W.-Y. Lin, W.-Y. Lee, and T.-P. Hong, ‘Adapting Crossover and Mutation Rates in Genetic Algorithms.’, *J. Inf. Sci. Eng.*, vol. 19, pp. 889–903, Sep. 2003.

[225] H. E. Aguirre and K. Tanaka, ‘Parallel Varying Mutation in Deterministic and Self-adaptive GAs’, in *Parallel Problem Solving from Nature — PPSN VII*, vol. 2439, J. J. M. Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, and J.-L. Fernández-Villacañas, Eds, in Lecture Notes in Computer Science, vol. 2439. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 111–121. doi: 10.1007/3-540-45712-7_11.

[226] S. Zhang, J. Xie, and H. Wang, ‘Fuzzy Adaptive NSGA-III for Large-Scale Optimization Problems’, *Int. J. Fuzzy Syst.*, vol. 24, no. 3, pp. 1619–1633, Apr. 2022, doi: 10.1007/s40815-021-01220-9.

[227] S. Tongchim and P. Chongstitvatana, ‘Parallel Genetic Algorithm with Parameter Adaptation’, *Information Processing Letters*, vol. 82, pp. 47–54, Apr. 2002, doi: 10.1016/S0020-0190(01)00286-1.

[228] C.-C. Hsu, S.-I. Yamada, H. Fujikawa, and K. Shida, ‘A fuzzy self-tuning parallel genetic algorithm for optimization’, *Computers & Industrial Engineering*, vol. 30, no. 4, pp. 883–893, Sep. 1996, doi: 10.1016/0360-8352(96)00039-3.

[229] Q. Cui, V. Rong, D. Chen, and W. Matusik, ‘Dense, Interlocking-Free and Scalable Spectral Packing of Generic 3D Objects’, *ACM Trans. Graph.*, vol. 42, no. 4, p. 141:1-141:14, Jul. 2023, doi: 10.1145/3592126.

[230] L. P. Tchapmi, C. B. Choy, I. Armeni, J. Gwak, and S. Savarese, ‘SEGCloud: Semantic Segmentation of 3D Point Clouds’, Oct. 20, 2017, *arXiv*: arXiv:1710.07563. doi: 10.48550/arXiv.1710.07563.

[231] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, ‘PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation’, Apr. 10, 2017, *arXiv*: arXiv:1612.00593. doi: 10.48550/arXiv.1612.00593.

[232] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, ‘PointNet++: deep hierarchical feature learning on point sets in a metric space’, in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, in NIPS’17. Red Hook, NY, USA: Curran Associates Inc., Dec. 2017, pp. 5105–5114.

[233] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, ‘ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes’, Apr. 11, 2017, *arXiv*: arXiv:1702.04405. doi: 10.48550/arXiv.1702.04405.

[234] K. Babacan, L. Chen, and G. Sohn, ‘SEMANTIC SEGMENTATION OF INDOOR POINT CLOUDS USING CONVOLUTIONAL NEURAL NETWORK’, *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. IV-4/W4, pp. 101–108, Nov. 2017, doi: 10.5194/isprs-annals-IV-4-W4-101-2017.

APPENDIX A

Data for results presented in Chapter 4, Section 4.4.4.

Free-Rotating Cuts

TABLE A–1. Data for low complexity, free rotating cuts. Green indicates optimum solution, yellow is suboptimal solution, red means failed to find a solution.

		First Feasible Random Search			Greedy Search			Height Decreasing Search			Stochastic Tree Search		
No. of Combinations	No. cut Parts	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)
24	4	34.00	1.000	1.29	34.00	1.000	0.50	34.00	1.000	0.38	34.00	1.000	0.71
120	5	39.50	0.962	2.20	38.00	1.000	0.69	38.00	1.000	0.32	38.00	1.000	1.21
720	6	59.20	0.980	4.44	58.00	1.000	0.99	61.00	0.951	0.50	58.00	1.000	9.13
5040	7	45.00	1.000	10.24	45.00	1.000	2.06	45.00	1.000	0.87	45.00	1.000	23.71
40320	8	63.00	1.000	18.67	63.00	1.000	1.34	77.00	0.818	0.38	63.00	1.000	145.28
362880	9	67.00	1.000	30.47	67.00	1.000	5.67	67.00	1.000	2.75	67.00	1.000	1929.22
3628800	10	92.00	0.957	49.42	0.00	0.000	0.00	94.00	0.936	0.80	88.00	1.000	3291.97
39916800	11	64.00	0.984	3.04	63.00	1.000	3.12	63.00	1.000	0.61	63.00	1.000	10800.00
Avg. Cost Strength			**0.985**			**0.875**			**0.963**			**1.000**	

TABLE A–2. Data for medium complexity, free rotating cuts. Green indicates optimum solution, yellow is suboptimal solution, red means failed to find a solution.

		First Feasible Random Search			Greedy Search			Height Decreasing Search			Stochastic Tree Search		
No. of Combinations	No. cut Parts	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)
2.09E+13	16	0.00	0.000	0.00	112.00	0.982	5.56	135.00	0.815	1.15	110.00	1.000	10800.00
3.56E+14	17	0.00	0.000	0.00	118.00	1.000	5.38	118.00	1.000	1.53	118.00	1.000	10800.00
6.40E+15	18	0.00	0.000	0.00	120.00	1.000	8.29	121.00	0.992	1.35	120.00	1.000	10800.00
2.43E+18	20	120.00	0.983	18.88	118.00	1.000	10.25	152.00	0.776	1.19	118.00	1.000	10800.00
5.11E+19	21	0.00	0.000	0.00	0.00	0.000	0.00	163.00	1.000	2.36	163.00	1.000	10800.00
6.20E+23	24	0.00	0.000	0.00	184.00	0.989	15.61	195.00	0.933	2.61	182.00	1.000	10800.00
3.05E+29	28	0.00	0.000	0.00	0.00	0.000	0.00	252.00	0.770	2.09	194.00	1.000	10800.00
Avg. Cost Strength			**0.140**			**0.710**			**0.898**			**1.000**	

TABLE A–3. Data for high complexity, free rotating cuts. Green indicates optimum solution, yellow is suboptimal solution, red means failed to find a solution.

		First Feasible Random Search			Greedy Search			Height Decreasing Search			Stochastic Tree Search		
No. of Combinations	No. cut Parts	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)
1.38E+43	37	0.00	0.000	0.00	277.00	1.000	39.76	301.00	0.920	3.92	277.00	1.000	10800.00
5.23E+44	38	0.00	0.000	0.00	273.00	1.000	34.12	322.00	0.848	3.22	274.67	0.994	10800.00
8.16E+47	40	0.00	0.000	0.00	313.00	0.995	40.00	334.00	0.932	3.50	311.33	1.000	10800.00
1.41E+51	42	0.00	0.000	0.00	346.00	1.000	50.51	414.00	0.836	4.19	348.67	0.992	10800.00
6.04E+52	43	0.00	0.000	0.00	287.00	1.000	44.94	396.00	0.725	4.92	287.00	1.000	10800.00
2.66E+54	44	0.00	0.000	0.00	303.00	1.000	48.58	364.00	0.832	5.30	303.00	1.000	10800.00
1.20E+56	45	0.00	0.000	0.00	0.00	0.000	0.00	375.00	0.904	5.45	339.00	1.000	10800.00
1.24E+61	48	0.00	0.000	0.00	378.00	1.000	58.51	436.00	0.867	5.57	379.00	0.997	10800.00
3.04E+64	50	0.00	0.000	0.00	370.00	1.000	82.87	386.00	0.959	8.55	374.00	0.989	10800.00
Avg. Cost Strength			**0.000**			**0.888**			**0.869**			**0.997**	

Orthogonal Cuts

TABLE A–4. Data for low complexity, orthogonal cuts. Green indicates optimum solution, yellow is suboptimal solution, red means failed to find a solution.

		First Feasible Random Search			Greedy Search			Height Decreasing Search			Stochastic Tree Search		
No. of Combinations	No. cut Parts	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)
24	4	19.00	1.000	0.16	19.00	1.000	0.21	19.00	1.000	0.21	19.00	1.000	0.21
120	5	26.00	1.000	1.97	26.00	1.000	1.14	26.00	1.000	0.57	26.00	1.000	2.09
720	6	43.00	1.000	3.13	43.00	1.000	1.27	43.00	1.000	1.32	43.00	1.000	1.21
5040	7	50.00	1.000	1.75	50.00	1.000	1.85	50.00	1.000	0.60	50.00	1.000	219.09
40320	8	60.00	1.000	5.06	60.00	1.000	1.91	60.00	1.000	0.47	60.00	1.000	1663.56
362880	9	105.00	0.648	4.73	68.00	1.000	5.01	113.00	0.602	0.80	68.00	1.000	10800
3628800	10	95.00	0.990	30.90	94.00	1.000	2.47	94.00	1.000	0.68	94.00	1.000	10800
39916800	11	92.00	1.000	18.26	92.00	1.000	3.04	106.00	0.868	0.68	92.00	1.000	10800
4.79E+08	12	107.00	1.000	26.91	107.00	1.000	4.87	107.00	1.000	0.76	107.00	1.000	10800
Avg. Cost Strength			**0.960**			**1.000**			**0.941**			**1.000**	

TABLE A–5. Data for medium complexity, orthogonal cuts. Green indicates optimum solution, yellow is suboptimal solution, red means failed to find a solution.

		First Feasible Random Search			Greedy Search			Height Decreasing Search			Stochastic Tree Search		
No. of Combinations	No. cut Parts	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)
1.31E+12	15	0.00	0.000	0.00	118.00	1.000	5.41	138.00	0.855	0.92	118.00	1.000	10800.05
2.09E+13	16	133.00	0.962	138.57	128.00	1.000	7.04	0.00	0.000	0.00	128.00	1.000	10800.02
3.56E+14	17	0.00	0.000	0.00	112.00	1.000	7.73	158.00	0.709	1.84	112.00	1.000	10800.05
6.40E+15	18	0.00	0.000	0.00	127.00	1.000	6.26	164.00	0.774	1.01	127.00	1.000	10800.03
1.12E+21	22	0.00	0.000	0.00	160.00	1.000	10.38	213.00	0.751	1.32	160.00	1.000	10800.04
1.55E+25	25	0.00	0.000	0.00	178.00	1.000	13.60	224.00	0.795	1.62	178.00	1.000	10800.06
1.09E+28	27	0.00	0.000	0.00	200.00	1.000	16.86	266.00	0.752	2.22	200.00	1.000	10800.05
Avg. Cost Strength			**0.137**			**1.000**			**0.662**			**1.000**	

TABLE A–6. Data for high complexity, orthogonal cuts. Green indicates optimum solution, yellow is suboptimal solution, red means failed to find a solution.

		First Feasible Random Search			Greedy Search			Height Decreasing Search			Stochastic Tree Search		
No. of Combinations	No. cut Parts	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)	Cost	Cost Strength	Mean time (s)
2.65E+32	30	0.00	0.000	0.00	243	1	24.39	264	0.9205	2.83	243	1	10800
2.95E+38	34	0.00	0.000	0.00	0	0	0	380	0.6947	2.45	264	1	10800
1.38E+43	37	0.00	0.000	0.00	255	1	34.58	400	0.6375	3.25	255	1	10800
5.23E+44	38	0.00	0.000	0.00	252	1	45.42	386	0.6528	4.34	272	0.9265	10800
3.35E+49	41	0.00	0.000	0.00	293	1	39.55	364	0.8049	3.89	293	1	10800
6.04E+52	43	0.00	0.000	0.00	340	1	33.54	388	0.8763	2.91	340	1	10800
2.59E+59	47	0.00	0.000	0.00	340	1	55.48	498	0.6827	6.52	349	0.9742	10800
8.07E+67	52	0.00	0.000	0.00	384	1	69.3	506	0.7589	5.98	387	0.9922	10800
Avg. Cost Strength			**0.000**			**0.875**			**0.754**			**0.987**	