Departamento de
Ingeniería Informática y de Sistemas

Doctoral Thesis

Multi-Level Filling Heuristic and an Instance Generator for the Multi-Objective 3D Packing Problem

Yanira González González

Supervisors:

Dra. Coromoto León Hernández

Dra. Gara Miranda Valladares

- 2017 -

Doctoral Dissertation

Multi-Level Filling Heuristic and an Instance Generator for the Multi-Objective 3D Packing Problem

Author

Yanira González González

Supervised by

Dra. Coromoto León Hernández

Dra. Gara Miranda Valladares

May 2017

Dra. **Coromoto León Hernández**, con N.I.F. 78.605.216-W, profesora Titular de Universidad, y Dra. **Gara Miranda Valladares**, con N.I.F. 78.563.584-T, profesora Ayudante Doctor, adscritas al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, y al área de Lenguajes y Sistemas Informáticos

I N F O R M A N

Que la presente memoria titulada:

"*Multi-Level Filling Heuristic and an Instance Generator for the Multi-Objective 3D Packing Problem*"

ha sido realizada bajo su dirección por Dª. **Yanira González González**, con N.I.F. 78.555.933-P, y constituye su Tesis para optar al grado de Doctor por la Universidad de La Laguna.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firmamos la presente en La Laguna, a 17 de mayo de 2017.

Acknowledgements

First of all, I wish to express my gratitude to my advisors Coromoto León and Gara Miranda, for their continued enthusiasm, dedication, effort and encouragement. Their support have been very important during these years. I would also like to thank my colleagues Jesica de Armas and Javier Villamonte, who have collaborated in this research. Also, I want to make mention to Eduardo Segredo for his technical help.

I owe special thanks to my parents, my sisters and my uncle, for their great patience, encouragement, endless support and to be source of motivation. I want to make a special mention to my sister, Yurena, who has had to read all my thesis. And in general to my family. Thank you. The way has been hard, however we are here.

Last but not least, I would like to thank my friends and workmates, especially dedicated to Jezabel and Alejandro, for their encouragement and support. I would also like to acknowledge Carlos (Chico:mente) for his art with the three-dimensional objects.

I am also grateful to CajaCanarias for grant by post-graduate, which made this research possible. This work was also supported by the EC (FEDER) and the Spanish Ministry of Science and Technology as part of the 'Plan Nacional de I+D+i', with contract numbers: TIN2016-78410-R, TIC2011-25448, and TIN2008-06491-C04-02. The Canary Government partially funded this work through the PI2007/015 research project. I also want to thank the HPC-EUROPA project (NUMBER 228398), supported by the European Commission - Capacities Area - Research Infrastructures Initiative.

Yanira González González

Preface

Cutting and packing problems belong to the family of combinatorial optimisation. When dealing with optimisation problems, the goal is to analyse and identify the alternative that most closely approximates the optimal solution. Finding an optimisation process that yields a truly optimal solution, however, is not a simple task. That is why finding solutions to optimisation problems continues to be a very active and dynamic area of research to this day. The interest in optimisation is closely linked to the search for alternatives to deal with problems in the everyday world, physical or material problems from the real world. Optimisation covers several areas of research in engineering, where a large part of the problems are part of complex systems for which there is no simple and general method for efficiently optimising either the problems or their possible solutions. Hence the need exists to constantly study and improve the optimisation processes. Many interdisciplinary factors are involved in the design of optimisation schemes, but primarily statistics, mathematics and computer science. Therefore, when applying these factors to analysing a specific problem, one has to consider all of the aspects from the field to which the problem belongs.

Depending on the number of objectives, optimisation problems can be classified into single-objective or multi-objective. Single-objective problems aim to optimise a single objective to be maximised or minimised. In this type of problem, the possible solutions are easy to compare, since it is just a matter of evaluating which solution is best for the objective in question. In the case of multi-objective optimisation problems, several objectives are optimised at once, which makes comparing the possible solutions an indirect process.

This work concerns itself with a study of the 3D Packing Problem, 3DPP. Cutting and packing problems have been studied in depth for numerous areas of industry and research. The 3DPP proposed in this work is of most concern in industry and in the transport of goods due to its relevance to a wide variety of real applications. When solving a problem of this type, the objective is normally to arrange a set of rectangular items (boxes) inside a rectangular object of larger dimensions (container) so as to maximise the volume of the cargo. However, there is one important aspect that the literature normally ignores when dealing with this type of problem, which is the tare limit that each type of container has. For example, the cost of renting lorries to transport goods is calculated based on the total weight that they can transport, independently of the cargo volume. It is thus beneficial to determine the loading pattern that allows maximising the cargo volume while at the same time maximising the value of the accumulated weight. Along these lines, the problem studied in this thesis is proposed as a Multi-Objective Optimization Problem (MOP), whose objective is not just to maximise the cargo volume, but also its weight inside the container.

The solution algorithms can be classified into two types: exact and approximate. Exact algorithms guarantee finding the best solution for the problem in question. They have the drawback, however, of being highly time and resource intensive. In contrast, approximate algorithms do not guarantee finding the optimal solution but they do have lower time and resource requirements. Approximate algorithms include heuristics and metaheuristics. Heuristics are ad hoc methods designed to solve a specific problem. They rely on concrete knowledge of the problem to yield high-quality solutions without requiring an excessive computational effort. Metaheuristics are more general methods that can be adapted to different problems and can better utilise the computational resources. These methods offer a good compromise between the effort needed to apply them and the quality of the solutions they yield. Computationally, the 3DPP problem is hard to solve, meaning that an exact solution cannot be obtained in polynomial time. Thus, although there are isolated works that approach it using exact algorithms, most studies focus on providing solutions that rely on heuristics and metaheuristics. The heuristics applied to the 3DPP must be developed taking into account different distributions of specific pieces. Recent years have seen an increase in the evaluation of metaheuristics to solve the 3D Packing Problem, such as genetic algorithms, simulated annealing algorithms, tabu search algorithms and hybrid algorithms. Specifically, the evolutionary algorithms have taken on great significance. They are a type of metaheuristics whose design is inspired by biological evolution and its genetic/molecular basis. Multi-Objective Optimization Evolutionary Algorithms (MOEAs) have shown real promise in solving real-world multi-objective problems. In particular, they have yielded competitive solutions for several cutting and packing problems. In this context, the difference between using a solution calculated quickly and using more sophisticated proposals to find the optimal solution can determine the very survival of a company. Developing these sophisticated and effective proposals, however, normally entails a significant computational effort that in real applications can result in reduced production speeds. It is thus essential that we find proposals that are both effective and efficient.

Most of the research into 3DPP consider the single-objective version of the problem or it weighs the different objectives. As a result, most of the existing test cases present in the literature are some variation of this model. When we study the same problem in its multi-objective formulation, however, there are no test cases that can be used to evaluate heuristics and metaheuristics, or to compare their results. The only set of data available in the literature is used by Derely et al. in their research, which employs one instance containing the characteristics of a set of 12 products by the Procter & Gamble distribution company. Said instance provides the quantity, dimensions and weight of each of the products. Using a single instance, however, is not sufficient to conduct a full computational study. A broad range of instances has to be generated for the research to be useful.

Thus, the goal of this work is to adapt the problem to a more practical version by incorporating objectives that turn the research into a case study that more closely resembles the reality of industry. To this end, we conducted a wide-ranging and detailed review of the literature on heuristics and metaheuristics applied to the 3DPP, so as to subsequently propose a multi-objective implementation. To our knowledge, there are no existing references where this is done. In summary, the main contributions of this work are:

- A review of the existing proposals to solve the problem.
- Implementation of a proposal to solve the problem from a multi-objective formulation by providing:
  - A coding scheme employed in positioning heuristics that indicates the sequence for arranging items and their orientations.
  - Different methods for evaluating the positioning of the items.
  - Different genetic operators.
- Development of software to generate test cases of varying complexities with identical boxes, weakly heterogeneous boxes and strongly heterogeneous boxes. This allows evaluating the robustness of the heuristics and optimisation models used.
- Development of software to display potential packaging patterns. This can be used to see the container loading process based on the solution generated by the optimisation model employed.

The general structure of the contents is described below. The first two chapters deal with the theoretical foundations on which this work is based. Chapter 1 introduces the concepts and solution methods involved in optimisation problems.

Chapter 2 gathers general information on cutting and packing problems, such as their typologies. This same chapter also details 3D cutting and packing problems, which are at the heart of the problem to be solved, 3DPP. To that end, the subsequent chapters describe the studies and research conducted involving the problem at hand. The multi-objective 3D Packing Problem is considered in detail in Chapter 3. To address the lack of test cases for the multi-objective 3DPP, Chapter 4 considers the development of an instance generator, showing the scheme for the instances generated. Chapter 5 presents a GUI that can be used to display the results generated by the different methods developed to solve the problem. Chapter 6 is devoted to the computational results. Lastly, Chapter 7 discusses the conclusions and in the Chapter 8 the future lines of work.

Contents

List of Algorithms

List of Figures

List of Tables

List of Acronyms

3DPP	Three-Dimensional Packing Problem
3D-MGL	Three-Dimensional Multi-Objective Generator in Length
3D-MGD	Three-Dimensional Multi-Objective Generator in Depth
API	Application Programming Interface
EA	Evolutionary Algorithm
BIM	Bit-Inversion Mutation
CLP	Container Loading Problem
CSS	Cascading Style Sheets
PP	3D Packing Problem
EC	Evolutionary Computation
ESICUP	Euro Special Interest Group on Cutting and Packing
EMS	Empty Maximum Spaces
ESICUP	Euro Special Interest Group on Cutting and Packing
GA	Genetic Algorithm
GPU	Graphics Processing Unit
GRASP	Greedy Randomised Adaptive Search Procedure
GUI	Graphical User Interface
GP	Genetic Programming
HCI	Human-Computer Interaction
HTML	HyperText Markup Language
IBEA	Indicator-Based Evolutionary Algorithm
IIPP	Identical Item Packing Problem
ILS	Iterated Local Search
LRO	Large Rectangular Object
MBSBPP	Multiple Bin-Size Bin Packing Problem
METCO	Metaheuristics-based Extensible Tool for Cooperative Optimisation

MILOPP	Multiple Identical Large Object Placement Problem
MIKP	Multiple Identical Knapsack Problem
MIP	Mixed Integer Linear Programming
MHKP	Multiple Heterogeneous Knapsack Problem
MHLOPP	Multiple Heterogeneous Large Object Placement Problem
MOEA	Multi-Objective Evolutionary Algorithm
MOGA	Multi-Objective Optimisation Genetic Algorithm
MLFHL	Multiple-Level Filling Heuristic in Length
MLFHD	Multiple-Level Filling Heuristic in Depth
MOP	Multi-Objective Optimisation Problem
MOSPP	Multi-Objective Strip Packing Problem
MPI	Message Passing Interface
MSSCSP	Multiple Stock-Size Cutting Stock Problem
NPGA	Niched Pareto Genetic Algorithm
NSGA	Non-Dominated Sorting Genetic Algorithm
NSGA-II	Non-Dominated Sorting Genetic Algorithm II
NSGA-III	Non-Dominated Sorting Genetic Algorithm III
ODP	Open Dimension Problem
ODP/W	Open Dimension Problem Weakly
ODP/S	Open Dimension Problem Strongly
OPX	One Point Crossover
PAES	Pareto Archived Evolution Strategy
PMOEA	Parallel Multi-Objective Evolutionary Algorithm
RBPP	Residual Bin Packing Problem
RCSP	Residual Cutting Stock Problem
SA	Simulated Annealing
SBX	Simulated Binary Crossover
SBSBPP	Single Bin-Size Bin Packing Problem
SI	Swarm Intelligence
SLFHL	Single-Level Filling Heuristic in Length
SLFHD	Single-Level Filling Heuristic in Depth
SKP	Single Knapsack Problem
SLOPP	Single Large Object Placement Problem
SPEA	Strength Pareto Evolutionary Algorithm
SPEA2	Strength Pareto Evolutionary Algorithm 2
SSSCSP	Single Stock-Size Cutting Stock Problem

TS	Tabu Search
UM	Uniform Mutation
UX	Uniform Crossover
VEGA	Vector-Evaluated Genetic Algorithm
VNS	Variable Neighbourhood Search
WEBGL	Web Graphics Library
XML	eXtensible Markup Language

Part I

Fundamentals and Background

Chapter

1 Optimisation

This chapter introduces the main concepts involving optimisation problems. First, we provide a formal definition of optimisation problems, introducing concepts such as the single-objective optimisation and multi-objective optimisation. We then offer a brief introduction to some of the strategies currently in use to tackle this class of problems in the literature, placing special emphasis on solving Multi-Objective Optimisation Problems (MOPs) and offering a summary of the techniques for solving MOPs that are best suited to solving problems like the one considered in this thesis, and specifically, to Multi-Objective Evolutionary Algorithms (MOEAs). We analyse both sequential and parallel techniques. Finally, an introduction to the parallel model based on dynamically-mapped islands is presented and employed, the goal being to obtain better solutions in less time.

1.1 Introduction

In their everyday lives, people experience a large range of situations in which, based on a set of possible alternatives, they must select one that offers the greatest benefit. In general, these situations, though typical, are not very complex and can be solved without having to resort to complex processes. Such is the case, for example, when deciding which path to take when going from one place to another in the shortest time possible. There are many other real-world problems, however, having to do with, for example, engineering, science, business and industry applications that cannot be resolved solely by using rational processes due to the complexity of the systems involved. Solving these problems requires developing processes that yield the optimum decision. For all of these reasons, and in an effort to find solutions to these problems, optimisation has turned into an important field of operations and computational research.

Optimisation is defined as the process that is carried out for the purpose of finding the best possible solution to an optimisation problem. In problems of this kind, the solution that yields the best of all feasible solutions is known as the optimal solution. In fields like computer science, artificial intelligence or operations research, optimisation has become an essential discipline for solving complex problems, thanks to its solid results. In most real problems, the optimal solution involves the available resources, limited time and the cost of the related processes. From a more formal point of view, an optimisation process aims to find the best solution from among a set of feasible solutions that satisfy a certain minimisation or maximisation function that is generally subject to restrictions. One example of this definition is the case of maximising the cargo volume inside a container considering as a constraint the vertical and horizontal stability of the cargo and the load bearing strength of the cargo [131]. Other restrictions may be considered for this type of problem, such as restrictions on orientation, load distribution or priority boarding [31]. In summary, the goal of optimisation is to analyse and identify the solution that best approaches the problem's optimal solution.

Many real-world engineering problems or decision-making problems consider the optimisation of more than one objective. These objectives are optimised simultaneously, which often leads to conflict, meaning that achieving the optimum for one objective requires some compromise on one or more other objectives. By way of example, consider a cargo delivery company that is strongly incentivised to maximise available cargo inside of the container and to minimise the number of trips across the global container transportation system [173].

Optimisation, as a powerful method to solve problems, has a wide range of applications in numerous fields [26]. Some possible examples of applications are those mentioned below. Optimisation is applied in container transportation and distribution industries [173], in cutting and packing problems [61], in communications and networks [193], in siting houses in residential neighbourhoods, in building layout, acoustic design, construction site layout [23], and in scheduling linear construction projects such as highways, pipelines and tunnels [13]. It is also considered in aeronautical design, from aerodynamic shape optimisation to structural design, trajectory optimisation, optimal control [138, 169], in computational optics in the field of planar integrated optics, optical communication technology, and dielectric material modelling [88], and in computational medicine [177]. The amount of related research in financial engineering and economics [179] and in some production industries [161] has also increased. All these problems try to find an optimal solution with respect to a certain objective function(s), considering specific constraints.

1.2 Optimisation Problems

Optimisation is being introduced for a wide range of fields in science, engineering and industry [219]. In these fields, optimisation is the process of analysing and identifying the best possible solution to solve a specific problem. Given an optimisation problem, there are various possible feasible solutions; however, the objective is to find the best possible solution. Hence, an optimisation problem consists of finding the best possible solution from among the set of feasible solutions by taking into account certain objectives, and subject to certain constraints that allow restricting the number of available alternatives [180]. A possible solution to an optimisation problem is determined by a series of decisions that depend on the features of the problem. For instance, fitting rectangular boxes into a container involves decisions such as determining how to stack the boxes, since those marked fragile will have to be placed at the top of the stack. Thus, solving an optimisation problem requires identifying and labelling the set of variables that depend on the problem domain. This set of variables is called the decision vector. A solution to an optimisation problem is generated by giving specific values to the decision vector by taking into consideration the defined constraints. The quality of a decision vector is determined by an evaluation function. Thus, an optimisation problem is given by [34]:

- A decision vector to represent the elements of the system to be modelled and that are controllable by the decider. The decisions depend on the problem's domain and these are assigned to each variable in the decision vector. The possible values taken by the decision variables affect the result of the optimisation/evaluation function.
- An optimisation/evaluation function to evaluate the quality of all possible feasible solutions.
- A set of constraints to restrict the possible values of the decision variables.

The set of all solutions that satisfy the restrictions makes up the so-called search space or solution space. Given a specific problem, the optimisation function can be to minimise or maximise. The optimisation function is called, variously, a loss or cost function, a fitness or utility function. When the goal is to minimise, the optimisation function is said to be a cost function (minimise effort). In other cases we speak of a fitness function (maximise benefit). Based on the optimisation function, optimisation problems can be classified as single-objective or multi-objective optimisation problems. In a single-objective optimisation problem, the goal is to find the minimum or maximum of a single optimisation function. In contrast, a multi-objective optimisation problem includes a set of optimisation functions, meaning that multi-objective optimisation often requires compromising conflicting goals [209, 216].

1.2.1 Single-Objective Optimisation Problems

In single-objective optimisation problems, the optimisation function f is a scalar function, and the goal is to find its optimal solution. An optimal solution is either its minimum or maximum value, depending on what the objective of the specific problem is. A single-objective optimisation problem is defined as [57, 209]:

Definition 1 *A general **single-objective optimisation problem** is defined as minimising (or maximising) $f(x)$ subject to $g_i(x) \leq 0$, $i = \{1, \ldots, m\}$, and $h_j(x) = 0$, $j = \{1, \ldots, p\}$ $x \in \Omega$. A solution minimises (or maximises) the scalar $f(x)$ where x is a n-dimensional decision variable vector $x = (x_1, \ldots, x_n)$ from some universe Ω.*

Note that a possible candidate solution should provide both constraints, $g_i(x) \leq 0$ and $h_j(x) = 0$, while optimising the objective function $f(x)$. Thus, $x = (x_1, \ldots, x_n)$ represents a decision vector, the values of which may be continuous or discrete whereas the objective function $f(x)$ can also be continuous or discrete. Ω contains the set of feasible solutions x that can be used to satisfy an evaluation of $f(x)$ and its constraints.

The method for finding the global optimum of an optimisation function is referred to as *Global Optimisation*, which is defined as [57]:

Definition 2 *Given a function $f : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, $\Omega \neq \emptyset$, for $x \in \Omega$ the value $f^* \triangleq f(x^*) > -\infty$ is called a **global minimum** if and only if*

$$\forall x \in \Omega : f(x^*) \leq f(x)$$

*x^* is by definition the global minimum solution, f is the objective function, and the set Ω is the feasible region of x. The goal of determining the global minimum solution(s) is called the **global optimisation problem** for a single-objective problem.*

1.2.2 Multi-Objective Optimisation Problems

Many real-world optimisation problems involve multiple objectives that must be simultaneously optimised. A Multi-Objective Optimisation Problem (MOP), also called a vector optimisation problem or multi-criteria, is defined as the problem of finding a vector of decision variables that satisfies a set of restrictions and optimises a vector function whose elements represent the objective functions [185, 202]. These functions comprise a mathematical description of the optimisation criteria. The objectives are often in conflict with one another and (at least partially) improving one of them could cause the other objectives to be affected, potentially reducing their quality. In other words, the values taken by the decision variables that optimise an objective could differ significantly from the optimal values for the other objectives. Thus, optimising a multi-objective problem requires finding a solution that would assign an acceptable value to all of the objective functions. Contrary to what happens in single-objective optimisation problems where the solution is usually clearly defined, MOPs present an uncountable set of solutions. There is thus no single solution that optimises every objective simultaneously. An MOP may be defined as [57]:

Definition 3 *A* ***multi-objective or multi-criteria optimisation problem*** *is defined as minimising/maximising* $F(x) = \{f_1(x), \ldots, f_k(x))\}$ *subject to* $g_i(x) \leq 0$, $i = \{1, \ldots, m\}$, *and* $h_j(x) = 0$, $j = \{1, \ldots, p\}$ $x \in \Omega$. *An* MOP *solution minimises/maximises the components of a vector* $F(x)$ *where* x *is an* n*-dimensional decision variable vector* $x = (x_1, \ldots, x_n)$ *from some universe* Ω. *Note that* $g_i(x) \leq 0$ *and* $h_j(x) = 0$ *represent constraints that must be fulfilled while minimising/maximising* $F(x)$ *and* Ω *contains all possible* x *that can be used to satisfy an evaluation of* $F(x)$.

Thus, an MOP consists of k objectives corresponding to the k objective functions, $m+p$ constraints on the objective functions and n decision variables. The k objective functions can be linear or non-linear and continuous or discrete. The representation of a potential solution and its corresponding interpretation yield the search space.

In practice, when dealing with MOPs, instead of a single optimum, there is a set of trade-off solutions formed by those solutions that are at least as good as others for every objective, and better for at least one of the objectives considered. Francis Ysidro Edgeworth (1881), and later Vilfredo Pareto (1896) [81], introduced the definition of the term optimal as the solution that provides acceptable values for every objective. In this kind of problem, an optimal solution is not identified, but rather a set of solutions called a *Pareto optimal* [55, 56, 57]. That is, a solution x is said to be non-dominated if there does not exist a solution y that performs better in at least one objective and at least as well as x in the rest. These solutions are optimal in the sense that no other solutions in the search space are superior to them when every objective is considered. They form the set of all solutions whose corresponding vectors are non-dominated with respect to all other comparison vectors [57]:

Definition 4 *A vector* $u = (u_1, \ldots, u_k)$ *is said to* ***dominate*** *another vector* $v = (v_1, \ldots, v_k)$ *(denoted by* $u \preceq v$*) if and only if* u *is partially less than* v : $\forall i \in \{1, \ldots, k\}$, $u_i \leq v_i \wedge \exists i \in \{1, \ldots, k\} : u_i < v_i$.

In optimisation there exists a function, called the evaluation function, that evaluates the quality of solution x from the decision space to the objective space. The evaluation function $F\colon \Omega \rightarrow \Lambda$ represents how, for each solution of the vector decision variables ($x = (x_1, \ldots, x_n)$) in the decision variable space, there is a point ($y = (y_1, \ldots, y_k)$) in the objective space. When the efficient solutions are represented in the objective space, they are collectively known as the Pareto front - also called the Pareto optimal front [57]. To restate, the Pareto optimal set is a subset of all possible solutions in Ω. Its evaluated objective vectors form the Pareto front, of which each vector is non-dominated with respect to all objective vectors produced by evaluating all possible solutions in Ω. The set that includes all the solutions from the Pareto optimal belonging to a region of the feasible space is called the *Pareto optimal set*, which is defined as [57]:

Definition 5 *For a given MOP, $F(x)$, the* ***Pareto Optimal Set****, $\mathcal{P}^*$, is defined as:*

$$\mathcal{P}^* := \{x \in \Omega \mid \nexists x' \in \Omega \quad F(x') \preceq F(x)\}$$

The final aim when dealing with MOPs is to obtain a non-dominated solution set that, in the best case, will coincide with the Pareto-optimal front.

Definition 6 *For a given MOP, $F(x)$, and Pareto Optimal Set, $\mathcal{P}^*$, the* ***Pareto Front*** *$\mathcal{PF}^*$ is defined as:*

$$\mathcal{PF}^* := \{u = F(x) \mid x \in \mathcal{P}^*\}$$

1.3 Heuristics

The optimisation algorithms used can be classified into two main categories: exact algorithms, and approximation algorithms [102]. Approximate algorithms include heuristics and metaheuristics. Exact algorithms are those that, with unlimited resources, are guaranteed to yield a solution or set of optimal solutions. These algorithms rely on an exhaustive exploration of the search space, and may enclose certain areas depending on the mathematical characteristics of the objective functions. The main drawback of these techniques is that they require large amounts of resources. Some exact methods, such as Branch and Bound [194], the A* Algorithm [203] and Dynamic Programming [49] have been used to solve problems with two objectives of limited size. However, even though the exact techniques are quite effective, their run-times are, in general, high, a drawback that can be compounded by problem instance sizes. As a result, it is often not viable to apply these kinds of techniques to many optimisation problems. With problems of a reduced scope and/or problems in which the decision making is to be conducted based on more than two objectives, it is common to resort to heuristic methods. As opposed to exact techniques, which ensure arriving at an optimal solution, heuristic techniques provide a good, but not necessarily optimal, solution in a reasonable computational time. This is where heuristic methods turn into efficient procedures for finding good solutions, even if proving their optimality is difficult. There are numerous sources that define the concept of heuristic. The one proposed by Thomas Weise [216] is:

Definition 7 *A heuristic is a part of an optimisation algorithm that uses the information currently gathered by the algorithm to help to decide which solution candidate should be tested next or how the next individual can be produced.*

Most real optimisation problems usually involve complex systems for which there is no simple and global method for optimising them efficiently. These highly complex and hard to solve problems are called NP-hard (*NP = Nondeterministic Polynomial*). Formally they are defined as problems that cannot be solved optimally or for which a better solution cannot be guaranteed using any exact or deterministic method within a reasonable polynomial time [34]. Thus, most real optimisation problems are usually solved using heuristic methods due to the inherent difficulty and their practical importance. In the literature, it is possible to find a large number of references that attempt to solve NP-hard optimisation problems. Thus, even though a heuristic does not guarantee optimality, it seeks an approximate solution within an acceptable computational time. A part of achieving good solutions in a reasonable computational time, there are other features that make heuristic methods effective, such as [156]:

- No exact method is known for solving the problem.
- There exists an exact method, but the computational time or resources are limited.
- Heuristic methods are more flexible than exact methods, and allow modelling difficult conditions.
- The heuristic method is used as part of a general process that ensures the optimal solution, either because the heuristic method provides a good initial solution, or because it participates in an intermediate step of the procedure.

In recent decades, there has been an increase in the development of heuristic procedures to solve optimisation problems. There are numerous heuristic methods, of various natures, that are highly correlated to specific problems. When dealing with heuristic algorithms, their development and ideas tend to be intrinsically related to the characteristics of the problem. Thus, a method designed for a specific problem will hardly be extrapolatable to another problem. In those cases where this can be done, the method must be specifically adapted to the new problem. As a result, it is difficult to establish a full classification for the heuristic methods, since more of them were designed for specific problems. In general, the best-known heuristic methods have been categorised into [156]:

- Decomposition methods, the original problem is broken down into sub-problems that are easier to solve.
- Inductive methods, the idea behind these methods is to generalise from smaller or simpler versions to the full case. The properties of techniques identified for these easier cases can then be applied to the full problem.
- Reduction methods, consist of identifying properties that are mostly satisfied by the good solutions and introducing them as problem restrictions. The objective is to restrict the space of solutions by simplifying the problem. There is the drawback of omitting optimal solutions from the original problem.
- Constructive methods, which require building the solution to the problem step by step. These are usually deterministic methods and are based on the best choice in each iteration. These methods have been widely used in classical combinatorial optimisation problems.
- Local search methods, these start with a feasible solution of the problem and improve it gradually. In each step of the procedure, the solution is shifted to one with a better value. The method ends when, for a given solution, there is no other accessible solution that improves on it.

1.4 Metaheuristics

Metaheuristics are widely used to solve many NP-hard optimisation problems. A metaheuristic is a type of approximate algorithm whose main idea relies on combining different heuristic methods at a higher level. The goal is to achieve a more effective exploration of the search space. In contrast to what happens with designing heuristics, a metaheuristic is a general-purpose algorithm. They are designed to solve complex problems without having to be adapted to each problem. Thus, metaheuristics tend to be iterative procedures that guide a subordinate search heuristic, intelligently combining different concepts to properly explore the search space. Formally, a metaheuristic is defined as [200]:

Definition 8 *A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimisation algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimisation algorithm according to the guidelines expressed in such a framework*

Both heuristic and metaheuristic techniques are search procedures that do not guarantee reaching the problem optimal. However, metaheuristics have mechanisms that allow them to escape from local optimal solutions as they attempt to reach the global optimal. This allows them to arrive at better solutions. Although metaheuristics do not guarantee optimality, and in general do not provide knowledge of the proximity of the solutions to the optimal, they do allow for the study of highly complex problems and yield good solutions in a reasonable computational time. These procedures are applied in a large number of fields: cutting and packing, loading and packing of containers, industrial processes, engineering, etc. Metaheuristic techniques have the following characteristics in common [34, 132]:

- Metaheuristics are strategies that guide the search process to efficiently explore the search and achieve near-optimal solutions.
- The techniques that constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually nondeterministic.
- Metaheuristics may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract-level description.
- Metaheuristics are not problem-specific and may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper-level strategy.

To obtain good solutions, any metaheuristic that is designed has to strike a suitable balance between intensification and diversification, these being opposing characteristics of the search process. Intensification refers the amount of effort employed to search within the current region (exploitation). Diversification is the amount of effort employed to search in different regions of the space (exploration). This balance is necessary to identify regions of the space with good-quality solutions without spending too much time in regions of the space that are not promising or that have already been explored.

There are different strategies for classifying metaheuristic, though there is no single rigorous or fully accepted classification. Depending on the characteristics used to define the classification, different taxonomies may result [34, 155, 210]:

- Based on the source of inspiration: nature-inspired, non-nature inspired.
- Based on the number of solutions: population-based search, single point search.
- Based on the objective functions: dynamic objective function, static objective function.

- Based on neighbourhood: one neighbourhood structure, various neighbourhood structures.
- Based on memory use: memory usage methods, memory-less methods.
- Based on strategy followed: constructive methods, trajectory-based, population-based.
- Based on type of procedure: relaxation methods, constructive processes, neighbourhood searches, evolutionary processes, decomposition processes, long-term memory processes.

Among the algorithms based on strategy followed are: trajectory-based and population-based. Trajectory-based metaheuristics start with a solution, which is updated by exploring the neighbourhood to create a trajectory. Population-based solutions are characterised by working with a set of solutions (population) in the search space in each iteration.

1.4.1 Trajectory-Based Metaheuristics

Trajectory-based metaheuristics are characterised by using a single solution while exploring the search space. This solution is updated as the neighbourhood is explored, with its quality improving in each iteration. During the process of reaching the optimal solution, a trajectory is traced out in the solution space, hence the name for these search strategies.

These algorithms arise as extensions of the local search strategies to which dispersion mechanisms are added, allowing them to escape from local minima. As a result, these require complex stopping conditions. The process is normally regarded as complete when the predefined upper limit for the number of iterations is reached, when a feasible solution is found or when stagnation is detected. Metaheuristics such as Simulated Annealing (SA), Tabu Search (TS), Greedy Randomised Adaptive Search Procedure (GRASP), Variable Neighbourhood Search (VNS) and Iterated Local Search (ILS) belong to this type of strategy.

1.4.2 Population-Based Metaheuristics

Population-based metaheuristics are characterised by the use of a solution set (population) with each iteration when exploring the search space. As a result, they may be regarded as an iterative improvement of the search process in which the population is initialised in the first iteration so that, after several iterations, a new population of solutions can be generated. The efficiency of this metaheuristic depends to a large extent on how the population is handled. The most widely-studied population-based methods are related to Evolutionary Computation (EC) and Swarm Intelligence (SI) [45]. EC algorithms are inspired by Darwin's theory of evolution, in which a population of individuals is modified through recombination and mutation operators. In SI, the idea is to reproduce the collective behaviour present in systems in nature [198] by studying the group dynamics exhibited by social animals. This social behaviour defines the motion of the decision variables in the search space, guiding them toward optimal solutions.

Research conducted on the principle of survival of the fittest observed in nature gave rise to computational simulations that proved very useful to solving complex problems, resulting in Evolutionary Algorithms (EAs).

1.4.2.1 Single-Objective Evolutionary Algorithms

Evolutionary Algorithms (EAs) are part of Evolutionary Computation (EC). EAs are population-based metaheuristic optimisation algorithms inspired by the idea of Darwinian evolution [59]. Darwin himself expressed how *"the environment ultimately selects individuals with the best suited genotypes to survive to reproduce. Those individuals who have more surviving offspring pass on more of their genes to the next generation"*. From an algorithmic point of view, there are many definitions for EA, one of which is [216]:

Definition 9 *Evolutionary Algorithms are population-based metaheuristic optimisation algorithms that use biology-inspired mechanisms like mutation, crossover, natural selection, and survival of the fittest in order to refine a set of solution candidates iteratively.*

The basic idea is to have a set of potential solutions to the problems, called individuals, referred to as a whole as a population, to which a series of genetic operators is applied in an effort to improve the best solutions found so far. The objective is to simulate the natural evolution of the species. By doing so, the individuals that best adapt to the medium they occupy have a higher probability of surviving and reproducing, thus passing their traits on to their offspring. In contrast, those individuals that do not adapt as well to the medium have a lower likelihood of surviving, and thus of reproducing, and will probably go extinct.

Considering the evolutionary model from its biological aspect reveals certain analogies to evolutionary algorithms. A chromosome, for example, consists of genes, each of which is responsible for an individual's traits. The information contained in chromosomes is called the genotype. When this information on the genotype is decided, it gives rise to a set of characteristics, the phenotype, that confer certain adaptability conditions upon the individual depending on the environment. Phenotypic traits increase the likelihood of reproduction, tending to increase in subsequent generations and providing a basis for new traits. If this is extrapolated to EAs, the counterpart of a chromosome is a chain of characters. Each of these characters is known as a gene, which correspond to the decision variables. This representation as a whole is known as a genotype. The counterpart to the phenotype is the solution that results from decoding this chain. Said decoding provides the input values used to evaluate the objective function.

In evolutionary algorithms, biological selection is simulated by a stochastic optimisation method. Every evolutionary algorithm consists of a population of individuals (chromosomes). Each member of the population (individual) is a candidate solution for the problem. Each individual is evaluated according a fitness function, objective function. Based on this evaluation, the quality of the solution is evaluated with respect to the population. Thus, an individual is evaluated as being some amount better or worse than the remaining individuals. The objective function is problem dependent. Furthermore, a set of genetic operators is applied to the population. These genetic operators introduce the two basic principles of biological evolution: selection and variation. Thus, evolutionary algorithms are population-based heuristic optimisation algorithms that rely on the theory of survival of the fittest to achieve a set of candidate solutions.

The evolutionary operators associated with evolutionary algorithms are classified into two main groups:

- Parent selection, which represents the struggle for survival. The goal is to select the set of parents that will take part in the reproduction process and create new individuals, offspring, within the population. These parents are selected based on their aptitude and their quality compared to the rest of the individuals. The selection method thus aims to select the best progenitors in the population to take part in the reproduction. The goal is to add pressure to improve the quality of the individuals that will go on to become part of the next generation.
- Variation operators are used to introduce new candidate elements to the population. These variation operators include the recombination, or crossover, operator and the mutation operator. Both operators are applied to the population's individuals using certain probabilities.
- Survivor selection or replacement operator, which preserves diversity in a population of individuals. The selection method decides which individuals among the parents and offspring survive to the next generation. Usually, offspring that survive replace weak individuals from previous population.

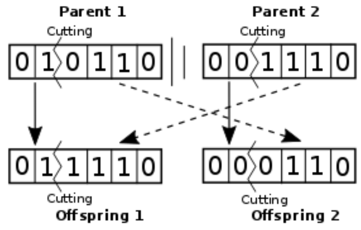


Figure 1.1: Generation of new individuals with One Point Crossover

The crossover operator allows groups of chromosomes (an encoded solution to the problem) to exchange fragments of their phenotypes (sub-chain of the chromosome that encodes a variable's value) in order to produce new chromosomes. Its purpose is to combine two or more parents to yield one or more children by taking the best characteristics of each parent. There is a wide variety of formulas in the literature for carrying out this exchange. The most common ones are: One Point Crossover (OPX), Simulated Binary Crossover (SBX) and Uniform Crossover (UX).

- The *One Point Crossover (OPX)* [122] involves exchanging genetic information between two parents by randomly selecting a cutoff point. This way, the information from the two progenitors is combined by taking the genetic information before the cutoff point for one parent and after said point for the other. This operation is shown in Figure 1.1.
- The *Simulated Binary Crossover (SBX)* [65] is used for crossovers with encoding based on actual numbers. This operator simulates the operation of the OPX operator when it is applied to binary string representations. The crossover is done such that some of the properties contained in binary crossover operators are maintained.
- The *Uniform Crossover (UX)* [205] may be viewed as a crossover of n points in which the number of crossover points is not specified beforehand; rather, it is randomly determined over the course of the operation. As a result, each gene has the same likelihood of being taken from either parent.

The mutation operator is used to introduce new genetic information into an individual at random. Its purpose is to allow for solutions to be generated that cannot be introduced by the crossover generator. This operator emulates the mutations that take place during a species' evolution when some of its genes are altered during the copying process. Mutations permit the population's diversity to increase. This

Algorithm 1 Generic Evolutionary Algorithm

1: **Initialisation.** Randomly generate the initial parent population.
2: **Evaluation.** Evaluate all individuals from the initial population by calculating the objective function.
3: **while** (stopping criterion is not satisfied) **do**
4: **Mating selection.** Select the individuals from the parent population to build the mating pool to generate the offspring.
5: **Variation.** Apply genetic operators to the mating pool to create a new offspring/child population.
6: **Evaluate.** Evaluate the offspring/child population generated.
7: **Survivor selection.** Select individuals for the next generation.
8: **end while**

operator is generally applied after the crossover operation. Some of the most widely used are: Bit-Inversion Mutation (BIM) and Uniform Mutation (UM).

- *Bit-Inversion Mutation (BIM)* [83] is used in binary representations by mutating each bit with a fixed probability.
- *Uniform Mutation (UM)* [83] is used for encodings based on real numbers. Each gene is mutated with a fixed probability. The new genetic value is assigned randomly in a uniform fashion from among the gene's acceptable values.

Although many EAs have been defined, they are all based on the same scheme [17, 34, 115, 191]. The Algorithm 1 shows the generic pseudocode for an EA. An evolutionary algorithm is initialised with a random population of N individuals - line 1. This initial population is evaluated - line 2 - by using the problem-dependent objective function. This assigns a fitness value to each individual that will determine its suitability as part of the problem's solution. The evolutionary loop is then entered. A series of steps must be carried out with each iteration or generation. First, the parents that are part of the mating pool - line 4 - are selected through the parent selection or mating selection mechanism. The mating pools are then subjected to variation operators to generate the offspring population. This introduces changes to the population. Once the offspring are generated, they are evaluated - line 6 - by the objective function so that they can be assigned a fitness value. Finally, a replacement or survivor selection operator is applied - line 7 - so as to determine the set of individuals (parents and children) that will survive to the next generation. The evolutionary algorithm ends when the stopping condition is reached - line 3.

1.4.2.2 Multi-Objective Evolutionary Algorithms

Multi-objective optimisation techniques seek to obtain a set of non-dominated solutions as close as possible to the Pareto front. Among the strategies used to solve

MOPs, Multi-Objective Evolutionary Algorithms (MOEAs) are amongst the most important. MOEAs have been shown to be suitable methods for tackling complex optimisation problems [82], which explains the increased use of multi-objective EAs in recent decades [64]. The efficiency of MOEAs for dealing with MOPs lies in their inherent parallelism and their ability to exploit similarities in solutions through recombination. This enables them to approach the Pareto front in a single execution. There are also elitist techniques that can be used to preserve non-dominated global solutions. Both single-objective and multi-objective EAs are based on the same algorithmic scheme. The Algorithm 1 shows the pseudocode that serves as the basis for defining this type of metaheuristics. The difference between them lies in how the fitness value is assigned and in the selection process. A single-objective EA contains a single optimisation function, whereas an MOEA calculates k ($k \geq 2$) optimisation functions.

Schaffer (1984) proposed the first implementation of what is now known as a MOEA [186]. It was with the Vector-Evaluated Genetic Algorithm (VEGA) [187], which lay the groundwork for what is today called a MOEA. There are currently multiple MOEAs in the literature, including: Multi-Objective Optimisation Genetic Algorithm (MOGA) [99], Pareto Archived Evolution Strategy (PAES) [137], Non-Dominated Sorting Genetic Algorithm (NSGA) [201], Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [67], Non-Dominated Sorting Genetic Algorithm III (NSGA-III) [66], Strength Pareto Evolutionary Algorithm (SPEA) [230], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [229], Indicator-Based Evolutionary Algorithm (IBEA) [228] and Adaptive Indicator-Based Evolutionary Algorithm (Adaptive-IBEA) [192]. Next, we describe some of the most commonly used in the literature, which are also those that have been used in this work.

Non-Dominated Sorting Genetic Algorithm II

Deb et al. [67] proposed a revised version of the NSGA [201], denoted as NSGA-II. This MOEA is based primarily on two principles. Use a fast non-dominated sorting approach, which makes the computation more effective. And keep a non-dominated elitist order that allows individuals with higher fitness to be preserved during the evolutionary process. The fast non-dominated approach, or selection operator, require defining an order for the individuals. The *crowded comparison operator* ($\geq_n$) is the component used to establish said order. This helps to reduce the complexity of the non-domination ordering procedure employed in the NSGA. This operator assigns two attributes to every individual i of the parent population: the *non-domination rank* ($rank_i$) and the *local crowding distance* ($distance_i$). Algorithm 2 shows the pseudocode of the NSGA-II [191]. It shares its basic scheme with the EAs, and is based on an initial population that will vary with each iteration of the algorithm.

Algorithm 2 Non-Dominated Sorting Genetic Algorithm II

1: **Initialisation.** Ramdomly generate the initial parent population P_t with N individivuals. Assign $t = 0$.
2: **Evaluation**. Evaluate all individuals in the initial parent population by calculating the objective functions.
3: **while** (stopping criterion is not satisfied) **do**
4: **Fitness assignment**. Calculate the fitness values of individuals in P_t. Use the non-domination rank in the first generation, and the crowded comparison operator in the remaining generations.
5: **Mating selection**. Perform deterministic binary tournament selection with replacement of P_t in order to fill the mating pool with N parents.
6: **Variation**. Apply genetic operators to the individuals of the mating pool in order to create the Child Population (CP) with $M = N$ new individuals.
7: **Evaluation**. Evaluate individuals in CP by computing the objective functions.
8: **Survivor selection**. Combine P_t and CP, selecting the best individuals using the crowded comparison operator to constitute P_{t+1}.
9: **end while**

The non-domination rank makes use of the *Pareto Dominance* concept. During the non-domination ranking process, the population is classified by fronts. The individuals belonging to the first front are non-dominated and are assigned the first rank. The individuals belonging to the second front are non-dominated in the absence of the previous front. The process then continues, considering only those individuals that do not have a rank assigned. Each individual is assigned a rank equivalent to its non-dominance level. The rank assigned in each step is increased by one. The process ends when every individual in the population is assigned its associated rank. This process ensures that the best solutions will be a part of the population in the next iteration.

The local crowding distance is used to estimate the density of solutions surrounding a particular individual. The greater the crowding distance the better, since this indicates a lower concentration of solutions in the region, thus preserving diversity. The procedure to calculate it is as follows [191]. First, the size of the largest cuboid enclosing the individual i without including any other individual that belongs to its rank is calculated. Then, the crowding distance is given by the mean side-length of the cuboid. It is important to note that the local crowding distance of the boundary individuals in every rank is assigned an infinite value. Finally, the partial order given by the crowded comparison operator $\geq_n$ is as follows:

$$i \geq_n j \ if \begin{cases} (rank_i < rank_j) \\ or \\ ((rank_i = rank_j) \ and \ (distance_i > distance_j)) \end{cases} \tag{1.1}$$

Strength Pareto Evolutionary Algorithm 2

SPEA2 is another algorithm that is widely studied in the literature. This MOEA was proposed by Zitzler et al. [229] and is an improvement on the SPEA [230]. This algorithm introduces elitism by using an external file in which to store the fittest individuals found. The SPEA2 incorporates a fine-grained fitness assignment strategy that allows ranking the individuals of the population. This strategy considers the number of individuals dominated by each solution and the number of individuals that dominate it. Thus, for each individual i, its fitness value is calculated as the sum of its strength raw fitness and a density estimate. The density information ($density_i$) allows eliminating individuals with identical raw fitness values, thus preserving the maximum range of non-dominated solutions. In order to calculate the raw fitness, the strength $strength_i$ of each individual i is calculated as the number of individuals that it dominates considering the parent population P_t and the file $\overline{P_t}$ [191]:

$$strength_i = \left| \left\{ j \mid j \in P_t \cup \overline{P_t} \wedge i \preceq j \right\} \right| \tag{1.2}$$

Then, the raw fitness raw_i of each individual i is calculated as the sum of $strength_j$ of all the individuals that dominate i. The calculation is as follows:

$$raw_i = \sum_{j \in P_t \cup \overline{P_t}, j \preceq i} strength_j \tag{1.3}$$

The SPEA2 features techniques for estimating density ($density$) that are used to guide the search more efficiently. These techniques allow avoiding repeated, or very similar, solutions on the Pareto front. The density estimator is an adaptation of the k-th nearest neighbour method [197]. Thus, for each individual i, its density estimator is calculated by:

$$density_i = \frac{1}{\sigma_k^i + 2} \tag{1.4}$$

where $\frac{1}{\sigma_k^i + 2}$ is the k-th element of a list arranged incrementally containing the distances from i to every other individual ($P_t \cup \overline{P_t}$). In other words, it corresponds to the distance between i and its k-th nearest neighbour.

Algorithm 3 shows the pseudocode for the SPEA2 [191]. The component N represents the population size, and $\overline{N}$ represents the desired file size. Both must be assigned by the user.

Algorithm 3 Strength Pareto Evolutionary Algorithm 2

1: **Initialisation**. Randomly generate the initial parent population P_t with N individuals, and create the empty file $\overline{P_t}$. Assign $t = 0$.
2: **while** (stopping criterion is not satisfied) **do**
3: **Evaluation**. Evaluate all individuals in the parent population by calculating the objective functions.
4: **Fitness assignment**. Calculate the fitness values of individuals in P_t and $\overline{P_t}$. For each individual i, calculate the raw fitness raw_i and the density estimate $density_i$
5: **Environmental selection**. Copy non-dominated individuals which belong to P_t and $\overline{P_t}$ to $\overline{P_{t+1}}$ If $|\overline{P_{t+1}}| > \overline{N}$ reduce $\overline{P_{t+1}}$ by means of the truncation operator. Otherwise, if $|\overline{P_{t+1}}| < N$, fill $\overline{P_{t+1}}$ with dominated individuals belonging to P_t and $\overline{P_t}$, considering their fitness.
6: **Mating selection**. Perform deterministic binary tournament selection with replacement on $\overline{P_{t+1}}$ to fill the mating pool with N parents.
7: **Variation**. Apply the crossover and mutation operators with probabilities p_c and p_m, respectively, to the mating pool so as to obtain $M = N$ offspring.
8: **Survivor selection**. Set P_{t+1} to the offspring population.
9: **end while**

Indicator-Based Evolutionary Algorithm

The IBEA is an evolutionary algorithm based on the indicator proposed by Zitzler and Künzli [228]. The main characteristic of this MOEA is that its fitness assignment scheme is based on a binary quality indicator. There are different binary quality indicators. One of them is the ϵ-indicator [232]. This algorithm is based on a random binary tournament selection scheme with replacement to generate new individuals from those selected solutions. The individuals with the lowest fitness are thus eliminated during the selection stage. This step is iterated until the population size does not exceed the initial population size N. Then, every time an individual is removed from the population, the fitness value must be recalculated for the remaining individuals still in the population. There are two versions of the IBEA, a basic one and a version known as adaptive. This latter version is much more robust. Algorithm 7 shows the pseudocode for the adaptive version of the IBEA [192]. In the Adaptive-IBEA, objective values are normalised and indicator values are adaptively scaled.

1.4.3 Parallel Metaheuristics

Most optimisation problems are processed sequentially, meaning the instructions are executed one after the other, such that only one instruction can be executed at any point in time. Due to the growing demand to solve complex, real-world problems with a high computational cost, the sequential model has certain disadvantages

Algorithm 4 Indicator-Based Evolutionary Algorithm (Adaptive Version)

1: **Initialisation**. Randomly generate the initial parent population P_t with N individuals. Assign $t = 0$.
2: **Evaluation**. Evaluate every individual in the initial parent population by calculating the objective functions.
3: **while** (stopping criterion is not satisfied) **do**
4: **Fitness assignment**. Calculate the fitness values using the quality indicator.
1. Calculate indicator values $I(x^1, x^2)$ using the normalised objective values $f_i^{'}$ and determine the maximum absolute value $c = max_{x^1,x^2 \in P}|I(x^1, x^2)|$
2. $\forall x^1 \in P, F(x^1) = \sum x^2 \in P \setminus \{x^1\}^{-e^{-I(\{x^2\},\{x^1\})/(c \cdot k)}}$
5: **Environmental selection**. While the size of P does not exceed N, remove the individual with the lowest fitness value, and recalculate the fitness value of the remaining individuals.
6: **Mating selection**. Perform binary tournament selection with replacement on P to fill the temporary mating pool $P^{'}$.
7: **Variation**. Apply the crossover and mutation with probabilities p_c and p_m, respectively, to the mating pool $P^{'}$ and add the resulting offspring to P.
8: **end while**

when it comes to finding optimal solutions. As a result, parallel models are being increasingly used to optimise problems. Parallel processing techniques allow, as their name indicates, executing two or more instructions in several processing units at the same time, thus reducing the length of time required to obtain high-quality solutions [16].

Parallel metaheuristics introduce the concept of computation speed. By dividing the workload amongst processors, they produce results in less time. In the case of exact algorithms, where the entire search space must be explored to ensure the optimal solution, the use of parallel computing considerably reduces execution times. Parallel schemes also reduce computation times in heuristics, but in some cases it is not reducing time that is of interest as much as accelerating the process to search for the best solution obtained in the sequential process.

When a MOP is solved using evolutionary algorithms, the first step is to select which algorithm to use. There is no guarantee, however, that the algorithm selected will be useful in solving the problem. If the algorithm chosen is not ideal for the problem at hand, it will most likely not find good solutions. Generally, this requires trying other algorithms. There is no a priori knowledge of which algorithm will behave best with a given problem, which could make the process very costly computationally. One alternative to lower the cost of these tests is to use island parallelisation techniques [123].

Island-based models have exhibited good performance and scalability in many areas [211]. In such a model, the population is divided into a number of independent sub-populations. Each sub-population is associated with an island and an evolutionary algorithm is executed for each sub-population. Each island evolves in isolation, though some solutions may occasionally be migrated between neighbouring islands. Different variants of island-based models have been proposed in the literature [48, 57].

- *Homogeneous island-based model*: every island executes the same MOEA with the same parameters, that is, each island executes one instance of the same algorithm.
- *Heterogeneous island-based model*: every island executes different MOEAs with different parameters, that is, each island executes one instance of a different algorithm.
- *Self-adaptive island-based, or team of algorithms, model* [95, 96]: each algorithm is weighted based on its behaviour during the execution. The best-behaving algorithm will be instanced in a larger number of islands, or, put another way, it will be allocated more of the available computational resources.

When dealing with a specific problem, if there is an MOEA that is clearly superior to the rest, a homogeneous island-based model that uses this algorithm will generally yield good-quality solutions. However, as noted, users normally have no existing knowledge of how the algorithm will behave when applied to a particular problem, thus resulting in the same problem that exists in the sequential case. The heterogeneous model can be used to execute different MOEAs and/or parameters in each processor at the same time, thus freeing the user from having to select a specific MOEA to solve the problem. If, however, one of the MOEAs included is not suited to optimising the problem, it wastes resources. To avoid this, one appropriate solution might be a model that attempts to assign resources to the most suitable optimisation method at any given time. This alternative comprises the so-called *teams of algorithms* [94, 95, 96] or *self-adaptive island-based model*. This technique allows the various algorithms to be dynamically weighted based on the results supplied for the problem at hand so that more computation time can be given to those that can achieve the best results.

In the standard island-based model, the configurations are assigned statically to the islands, meaning that each island executes the same configuration over the course of the execution. In the dynamic or adaptive assignment model, said assignment relies on a scoring strategy and on a selection strategy. Parallel execution models based on dynamically assigned islands consist of a master, or coordinating, island and as many execution, or slave, islands as required. The coordination process selects the configurations to be executed on the islands at all times based on the selection method indicated and initiates the executions in each of the idle slave processes. Each of the slaves comprises an execution island that is in charge of successively executing the various configurations specified by the coordinator.

Chapter 2

3D Cutting and Packing

This chapter provides a general overview of cutting and packing problems. Given the impact that these problems have on industry and on the diversity of practical applications they encompass, in this chapter we describe the basic properties that can be used to identify and differentiate between different types of cutting and packing problems. The rest of the chapter focuses on a real-world three-dimensional cutting and packing problem: container loading problem. Finally, we present a blueprint with all of the possible practical restrictions exhibited by loading problems in the literature.

2.1 Introduction

Cutting and Packing (C&P) encompasses a set of natural combinatorial optimisation problems. The cutting and packing problems have been widely studies over the last two decades in the industrial sector and research areas [134, 220]. These problems arise in a considerable variety of industrial applications where a set of small items must be loaded into one or more large objects, or where a large object must be cut into smaller items. Cutting and packing problems describe patterns made up by geometric combinations of small items assigned to large objects such that a small item is assigned to a single large object (Figure 2.1). Due to the important role played by patterns, and since geometric combinations are involved, it may be said that cutting and packing problems belong to the field of geometric combinatorics [75]. The cutting and packing problems consider the geometric objects and items or the space occupied by them, which is why these problems refer to objects and items defined into the three spatial dimensions of Euclidean space. In the case of cutting problems, they involve a solid material which must be cut into smaller items. For example, metal, glass, textiles, plastics, leather and wood. Packing and

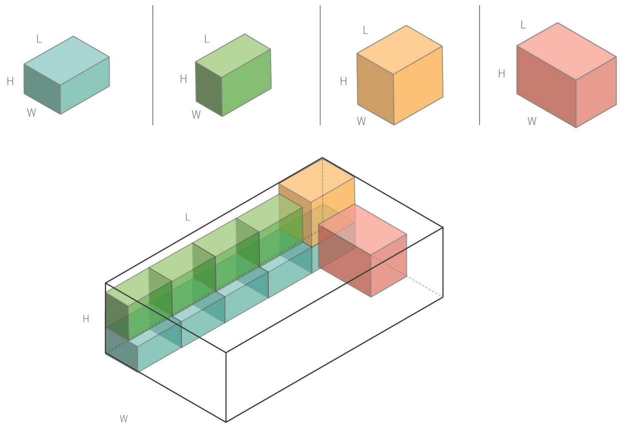


Figure 2.1: Example of loading and packing pattern

loading problems deal with three-dimensional large, empty objects like containers, trucks, bins, and so on. These empty spaces have to be loaded with small items while attempting to maximise the amount of space used. In this way, packing problems, such as the loading of containers, may be regarded as a cutting problem where the empty space of the large object must be cut into areas that can be occupied by small items. In other words, the strong relationship that exists between cutting and packing problems stems from the duality of material and space; that is, the duality between a solid material and the space it occupies. Because of this, cutting and packing problems may be referred to interchangeably. These problems not only consider the spatial dimensions of the objects and items, but also other non-spatial dimensions such as profit or weight, time and financial aspects [76].

Cutting and Packing problems have an identical logical structure in common. All of them consist of two sets of elements: a set of large objects (input, stock) and a set of small items (output, demand). The purpose of these problems is to select some or all of the small items, group them into one or more subgroups and assign each of the resulting subgroups to one of the large objects such that the geometric conditions are maintained (combinatoric base condition). Within each large object, the small items are to be arranged such that:

- all small items of the subset lie entirely within the large object,
- the small items do not overlap, and

- a given single- or multi-dimensional objective function is optimised.

The space of the large object that is not filled by the small items is referred to as *trim loss* (geometric base condition). Thus, based on the above, it is possible to define a set of different subproblems that have to be solved simultaneously in order to achieve a "global" optimum [215], such as:

- selection problem involving the large objects,
- selection problem involving the small items,
- grouping problem involving the selected small items,
- allocation problem involving the assignment of the subsets of the small items to the large objects, and
- layout problem involving the arrangement of the small items in each of the selected large objects with respect to the geometric condition.

When dealing with cutting and packing problems, it is normal to find restrictions that must be considered in order to solve the problem. These constraints are particularly relevant for a specific formulation of the problem.

2.2 Typology

The term cutting and packing has rapidly spread to many areas of research. In many of these disciplines, cutting and packing are used synonymously. This simplification is due to the strong relationship between cutting and packing, so it is usual to apply similar considerations to both problem categories [75]. In many cases this has resulted in problems with the same logical structure appearing in the literature under different names [75], such as:

- cutting stock and trim loss problems,
- bin packing, dual bin packing, strip packing, vector packing, knapsack (packing) problems, loading, pallet loading, container,
- vehicle loading, and car loading problems,
- assortment, design, dividing, layout, nesting, and partitioning problems.

Wäscher et al. define typology as *"a systematic organisation of objects into homogeneous categories on the basics of a given set of characterising criteria"* in [215]. Several attempts have been made in the literature to define typologies that can be used to classify cutting and packing in a general fashion. Some of these proposals are given in [73, 75, 121, 151, 184, 215]. A typology helps to standardise definitions and notations, thus facilitating the categorisation of cutting and packing problems in research, providing the foundation for a structural analysis of the different types of problems.

Thus, in an effort to classify cutting and packing problems, Dyckhfoff (1990) proposed a consistent and systematic approach for a comprehensive typology based on the logical structure of the problems [75]. Dyckhoff analysed various kinds of problems and identified nine main characteristics and elementary types among them [75]: dimensionality, quantity measurement, shape of figures, assortment, availability, pattern restrictions, assignment restrictions, objectives and status of information and variability. This typology has not been widely accepted in the field of cutting and packing however.

Proposing a suitable typology that can be used as a basis for the organisation and categorisation of existing and new literature entails analysing cutting and packing problems exhaustively to identify the main characteristics of each [215]. Normally, a logical structure in common is identified, which can be summarised as follows [75]:

- a set of large objects (input, supply) and
- a set of small items (output, demand),
- the patterns as geometric combinations of small items for one large object,
- the assignments of small items to patterns and of patterns to large objects.

However, there is still no internationally accepted typology for the class of cutting and packing problems. Wäscher et al. proposed a typology that improves on that originally proposed by Dyckhoff [75] attempting to address several of its limitations, such as [215]:

- In some cases the same problem can be classified under different notations.
- Two different problems that are solved differently can be similarly classified.

So, an improved typology is presented that can be used to attribute common properties to apparently heterogeneous problems and to recognise general similarities. This typology comprises five main characteristics to define problem types: dimensionality, kind of assignment, assortment of small items, assortment of large objects, and shape of the small items. Each of these is described briefly below.

Dimensionality

This property is an attributed to the problem or, more precisely, to the patterns. The dimensionality is the minimum number of dimensions of real numbers necessary to describe the geometry of the patterns [75]. We may distinguish between one-, two-, three-, and n-dimensional problems ($n > 3$).

Kind of assignment

In this case, two different problem types are introduced, depending on how the small items are assigned to the large objects [215].

- Output (value) maximisation: a set of small items is assigned to a given set of large objects. The set of large objects is not sufficient to accommodate all of the small items. All large objects are to be used; that is, there is no selection problem in term of the large objects. The selection involves the small items, the goal being to maximise the assignment value.
- Input (value) minimisation: a given set of small items must be assigned to a set of large objects. In contrast to the preceding case, the set of large objects is sufficient to accommodate all of the small items. Thus, a subset of large objects must be selected in which to accommodate all of the small items while minimising losses. In other words, the small items must be assigned to a selection of large objects of minimum value. In this case, there is no selection problem involving the small items.

Assortment of small items

With respect to the assortment of small items, we may distinguish among the next three cases [215]:

- Identical small items: every item has the same shape and size. In some output (value) maximisation formulations, it can be assumed that the single item type has an unlimited demand.
- Weakly heterogeneous assortment: the items can be grouped into relatively few classes - in relation to the total number of items -, in which the items are identical with respect to shape and size. By definition, small items which require different orientations are treated as different kinds. The demand for each item type is relatively large, and may or may not be limited.
- Strongly heterogeneous assortment: only very few elements have an identical shape and size. If that occurs, the items are treated as individual elements. Therefore, the demand for each item is equal to one.

Assortment of large objects

As concerns this characteristic, the following cases can be distinguished based on the number of available large objects [215]:

- One large object: the set of large objects consists of a single element. The range of the large object may be fixed in all problem-relevant dimensions, or its range may be variable in one or more dimensions.

- Several large objects: in this case all of the dimensions are assumed to be fixed when more than one large object is available. Thus, with respect to the assortment of large objects, we may distinguish between identical large objects and a weakly or strongly heterogeneous assortment of large objects.

When dealing with basic problem types in two or three dimensions, all large objects are assumed to have a rectangular shape (rectangles, cuboid) and to consist of a homogeneous material.

Shape of the small items

A figure may be described based on its form, size and orientation [160]. For the definition of refined problem types, we may distinguish between regular small items and irregular - non-regular - small items in terms of the shape of the objects. In most cases, the small items have regular shapes, such as rectangles, circles, boxes, cylinders, balls or blocks. The size of the objects is given by their length along each dimension. With respect to the orientation of the items, the next conditions might apply: any orientation is allowed, only 90-degree is allowed, or the orientation is fixed [160].

2.3 Classification

The typology introduced by Wäscher, Hauβner and Schumann presents the most general criteria for defining a cutting and packing problem. These criteria have provided a basis for classifying problems. However, when key criteria such as dimensionality, assignment and assortment of large and small items are combined, they yield a large number of different classes of cutting and packing problems. Thus, in an effort to provide a more general classification of the standard problems, their basic properties have been studied, from the most general to the most specific. This way, the basic types of cutting and packing problems result from combining criteria involving dimensionality, kind of assignment, assortment of small items, and assortment of large objects [215].

In the area of loading problems, and by combining the different criteria, we may distinguish between loading problems in which there are enough available large objects in which to introduce all of the small items, and problems in which only a subset of the small items can be packed into a fixed set of large objects [43]. Based on the assignment criteria, problems of first kind are of the input (value) minimisation type, and the problems of the second type are of the output (value) maximisation type. It is also possible to distinguish loading problems with respect to the assortment of small items and the assortment of the large objects. Small items belong to the same type if they are identical with regard to their shape and dimensions. The assortment of small items is considered as weakly heterogeneous when there are many items but a few item types. If there are few items and many item types, this is known as strongly heterogeneous. These same criteria are used with the large objects. When there is more than one large object, these can likewise be classified as identical, weakly or strongly heterogeneous.

2.3.1 Input Minimisation

Problems of the input minimisation type are characterised by the fact that the supply of large objects is large enough to accommodate all of the small items [215]. The aim, then, is to minimise the cost of the large objects needed to contain all of the the small items. Figure 2.2 shows the input (value) minimisation assignment problem types. Problems of this type are:

Cutting Stock Problem

Problems in this category require that a weakly heterogeneous assortment of small items be completely allocated to a selection of large objects of minimal value, number, or total size [215]. The size of the large objects is fixed in all dimensions, though with regard to their assortment, the following subcategories can be distinguished [44]:

- Single Stock-Size Cutting Stock Problem (SSSCSP) - packing a weakly heterogeneous set of small items into a minimum number of identical large objects.
- Multiple Stock-Size Cutting Stock Problem (MSSCSP) - packing a weakly heterogeneous set of small items into a weakly heterogeneous assortment of large objects such that the value of the large objects used is minimised.
- Residual Cutting Stock Problem (RCSP) - packing a weakly heterogeneous set of small items into a strongly heterogeneous assortment of large objects such that the value of the large objects used is minimised.

Bin Packing Problem

The Bin Packing Problem is characterised by a strongly heterogeneous assortment of small items. These small items have to be assigned to a set of identical large objects, a weakly or strongly heterogeneous assortment of large objects. The value, number, or total size of the large objects needed has to be minimised [215]. The problems include [44]:

- Single Bin-Size Bin Packing Problem (SBSBPP) - packing a strongly heterogeneous set of small items into a minimum number of identical large objects.

characteristics of large objects		assortment of small items: weakly heterogeneous	assortment of small items: strongly heterogeneous
all dimensions fixed	identical	Single Stock-Size Cutting Stock Problem (SSSCSP)	Single Bin-Size Bin Packing Problem (SBSBPP)
	weakly heterogeneous	Multiple Stock-Size Cutting Stock Problem (MSSCSP)	Multiple Bin-Size Bin Packing Problem (MBSBPP)
	strongly heterogeneous	Residual Cutting Stock Problem (RCSP)	Residual Bin Packing Problem (RBPP)
one large object variable dimensions(s)		Open Dimension Problem (ODP)	
		(ODP/W)	(ODP/S)

Figure 2.2: Input minimisation for basic types of cutting and packing problems

- Multiple Bin-Size Bin Packing Problem (MBSBPP) - packing a strongly heterogeneous set of small items into a weakly heterogeneous assortment of large objects such that the value of the large objects used is minimised.
- Residual Bin Packing Problem (RBPP) - packing a strongly heterogeneous set of small items into a strongly heterogeneous assortment of large objects such that the value of the large objects used is minimised.

Open Dimension Problem

The Open Dimension Problem (ODP) defines a problem category in which the set of small items has to fit completely into one or several large objects [215]. The large objects are given, but their size in at least one dimension can be considered variable. In general, most open dimension problems consider only one available large object. Thus, a grouping of cargo has to fit completely into a single container with one or more variable dimensions such that the container volume is minimised [44]. There are two variations of this problem type [44]:

- Open Dimension Problem Weakly (ODP/W) - problems with a weakly heterogeneous assortment of cargo.
- Open Dimension Problem Strongly (ODP/S) - problems with a strongly heterogeneous assortment of cargo.

2.3.2 Output Maximisation

All problems of the output (value) maximisation type have in common that the large objects are not sufficient to accommodate all of the small items. All of the large objects will be used, since the value of the items accommodated has to be maximised. In other words, there is generally a selection problem involving the small items, but none regarding the large objects. According to Figure 2.3, the following problem types may be defined [215]:

Identical Item Packing Problem

The Identical Item Packing Problem (IIPP) consists of assigning the largest possible number of identical small items to a given, limited set of large objects. Due to the fact that all of the the small items are identical, the problem is reduced to one of layout and involves the arrangement of identical small items in each of the large objects with respect to the geometric condition [215]. As for the loading problems, this problem category involves loading a single large object with a maximum number of identical small items [44].

Placement Problem

In the literature, problems of this category are known by many different names. In this case, this category describes problems in which a weakly heterogeneous assortment of small items has to be assigned to a given, limited set of large objects. The value or the total size of the small objects contained has to be maximised, or alternatively, the corresponding waste has to be minimised [215]. Based on how the large objects are assorted, the following problem types may be defined [44]:

- Single Large Object Placement Problem (SLOPP) - loading a single large object with a selection from a weakly heterogeneous set of small items such that the value of the loaded items is maximised.
- Multiple Identical Large Object Placement Problem (MILOPP) - loading a set of identical large objects with a selection from a weakly heterogeneous set of small items such that the value of the loaded items is maximised.
- Multiple Heterogeneous Large Object Placement Problem (MHLOPP) - loading a (weakly or strongly) heterogeneous set of large objects with a selection from a weakly heterogeneous set of small items such that the value of the loaded items is maximised.

Knapsack Problem

This problem represents a problem category that is characterised by a strongly heterogeneous assortment of small items which have to be allocated to a given set

		assortment of small items		
characteristics of large objects		identical	weakly heterogeneous	strongly heterogeneous
all dimensions fixed	one large object	Identical Item Packing Problem (IPP)	Single Large Object Placement Problem (SLOPP)	Single Knapsack Problem (SKP)
	identical		Multiple Identical Large Object Placement Problem (MILOPP)	Multiple Identical Knapsack Problem (MIKP)
	heterogeneous		Multiple Heterogeneous Large Object Placement Problem (MHLOPP)	Multiple Heterogeneous Knapsack Problem (MHKP)

Figure 2.3: Output maximisation for basic types of cutting and packing problems

of large objects. The availability of the large objects is fixed, and not all of the small items can be loaded. The value of the items accommodated has to be maximised [215]. Based on how the large objects are assorted, the following subproblems may be defined [44]:

- Single Knapsack Problem (SKP) - loading a single large object with a selection from a strongly heterogeneous set of small items such that the value of the loaded items is maximised.
- Multiple Identical Knapsack Problem (MIKP) - loading a set of identical large objects with a selection from a strongly heterogeneous set of small items such that the value of the loaded items is maximised.
- Multiple Heterogeneous Knapsack Problem (MHKP) - loading set of (weakly or strongly) heterogeneous large objects with a selection from a strongly heterogeneous set of small items such that the value of the loaded items is maximised.

2.4 3D Packing Problem

The 3D Packing and Loading Problem is a three-dimensional packing problem with numerous applications in the real world. In the transport sector and distribution industries, the most commonly cited applications are container loading and truck loading. Among the packing and loading problems, container loading problems have been highly studied. In [44], Bortfeldt and Wäscher define these problems as *geometric assignment problems, in which small, three-dimensional items (called cargo) have to be assigned (packed in) to three-dimensional, rectangular (cubic), large* *objects (called containers) such that a given objective function is optimised*, i.e. a set of rectangular pieces (boxes) are to be distributed in one large rectangular object (container) so as to maximise the total volume of packed boxes, while satisfying certain constraints. A solution to an assignment problem of this kind will be called a *loading pattern*.

2.4.1 Constraints

The packing and loading problem is borne of a necessity to improve the real-world scenarios in the industrial and commercial applications for global logistics. As a result, the packing and loading problem is often subject to a wide variety of constraints. In [31] many practical requirements, such as restrictions on orientation, load distribution, stability, priority boarding or the maximum weight of containers, are described. Sometimes, packing all of the small items in a large object requires observing constraints involving their orientation, for example when their content is fragile. Other times, the load must be correctly distributed in the large object for proper transport, or a specific order is required for subsequent shipment. However, a rather common aspect in the scope of this problem is the weight limit of the large objects, since they normally cannot exceed a certain transport weight. The goal then is to maximise the amount shipped without exceeding that limit. In particular, in loading problems, the set of small items represent boxes, whereas the large object refers to the container.

As a starting point, Bortfeldt and Wäscher provide a comprehensive list of real-world constraints of practical relevance to container loading and introduce a scheme for categorising them [44]. They suggested a classification with the more relevant constraints that can be encountered in the real world, and that are considered in the literature. They distinguish between constraints related to the large objects (container-related), to the small items as individual items (item-related), and to the entire or subset of items (cargo-related). They also consider constraints that can be related to the relationship between the large objects and the small items (positioning), and finally, the constraints involving the packing processes (load-related) [44]. These constraints may or may not be considered. There are, however, three basic constraints to the container loading problem that have been considered in most of the studies conducted:

- All of the small items lie entirely inside the large object.
- The small items cannot overlap.
- Small items only be located with their edges parallel to the walls of the large object.

The constraints can be presented as hard or as soft constraints. Hard constraints must always be satisfied, whilst soft constraints should be considered, allowing a certain margin of failure. In contrast, if a loading pattern does not satisfy all of the hard constraints, then that pattern is not feasible.

2.4.1.1 Large Object-Related Constraints

Weight limits

Weight limits determine the maximum weight that can be loaded inside the container. Every container has a weight limit that cannot be exceeded, denoted as its tare. Hence, small items can only be loaded inside the container as long a certain weight is not exceeded. In some cases, weight capacity constraints may be more restrictive than the available loading volume, which is an intrinsic constraint to the problem. Terno et al. [207], Gehring and Bortfeldt [106], Chan et al. [51], Bortfeldt et al. [40], Egeblad et al. [78], Liu et al. [149], Dereli and Das [70] are some of the works where the weight limit is addressed as a hard constraint.

Weight distribution

Weight distribution constraints, or load balance constraints, require that the weight of the cargo be spread out over the container floor. In transportation logistics, it is important for the cargo to be properly distributed. A balanced cargo can reduce the risk that the cargo will be damaged while the container is moved. In the literature, weight distribution constraints are often presented as soft constraints. In papers such as [32, 38, 78], the centre of gravity of the cargo is regarded as the geometric mid-point of the container floor. Other authors, such as Balakisky et al. [25], Liu et al. [149] pay special attention to the weight distribution along the container height, and consider the centre of gravity as being located as close to the floor as possible. Thus, heavier items should be stowed near the bottom and lighter items should be placed at a higher level.

2.4.1.2 Item-Related Constraints

Loading priorities

Loading priority constraints arise in loading problems of the output (value) maximisation type, where the available container volume is not large enough to accommodate all of the small items, and a decision has to be made regarding which items should be loaded and which will have to be excluded from the container. Sometimes it is more desirable to give priority to certain items, whether because of delivery deadlines or due to requirements related to the shelf life of products [29, 181]. When a subset of items must be loaded inside the container, we speak of a hard constraint. In contrast, when a subset of items is not considered a priority, this gives rise to a soft constraint. In papers such as Junqueira et al. [131], the loading priority is considered as an important constraint.

Orientation

Orientation constraints determine the possible orientation of a box. It is one of the most widely used constraints in the literature. Each item has six possible orientations in which it can be placed inside the container. However, in many cases, the number of orientations of an item may be restricted. Normally, the vertical orientation is limited to one dimension [100, 108, 127]. This restriction allows 90-degree rotations of the boxes in the horizontal plane. One example of this is the case in which the items have printed on them instructions like *"This way up!"*, requiring them to rest on a particular surface. In [22, 157] only a single orientation is allowed for each item in the vertical and horizontal directions. The opposite occurs in [79, 174], which has no restrictions with respect to the orientation of the items, meaning they can be freely rotated.

Stacking

Stacking constraints, or load-bearing constraints, restrict the possibility of placing items atop one another so as to avoid deforming them. These are thus related to the weight that an item can support without being damaged. Defining the way in which items can be stacked depends on each item's load-bearing strength. In the literature there are several ways to deal with this constraint. Junqueira et al. [131] and Bischoff [28] limit the number of items that can be stacked. In practice, the instruction *"Stack no more than x high"* is one example of this. Other approaches prohibit placing a particular type of item on top of another type [190, 207]. Larger items cannot be stacked atop smaller ones. However, the most usual stacking constraint is represented by the maximum weight an item can withstand without being damaged. This approach has been studied by Alonso et al. [20], Christensen et al. [54], and others.

2.4.1.3 Cargo-Related Constraints

Complete shipments

The need to have a set of items loaded together is known as the complete-shipments constraint. This class of constraint only makes sense when dealing with problems of the output (value) maximisation type, where the container's loading capacity is limited. This implies that the cargo must be accommodated efficiently. There are some papers in which if an item of one subset is loaded, then the remaining items of that same type must also be loaded in the container [85]. In some cases, when more than one container is used, it is not necessary to load all the items of the same subset into the same container, but they must be included in the same shipment. There are more restrictive cases, such as those considered by Iori et al. [127] and Moura et al. [164], which require all items of the same type to be loaded into the same container. In these cases, all of the components of a product - pieces of furniture - must be loaded together since they are going to the same destination.

Allocation

Allocation constraints arise in problems involving multiple containers. These problems may require packing items of a specific subset of the cargo inside the same container. Problems such as this one involve connectivity constraints, as in Liu et al. in [149]. In other, less restrictive, cases, certain items or types of items cannot be loaded into the same container, giving rise to separation constraints. In [85], Eley proposes a simple heuristic involving this type of constraint.

2.4.1.4 Positioning Constraints

Positioning constraints restrict the positions items in the container. These constraints may be given in absolute or relative terms. Absolute positioning constraints determine which items will be placed (or not) in a given position or area inside the container. This restriction is present in cases involving bulky objects, for example, in which for logistical reasons, these objects tend to be located near the doors. There are some studies, such as those by Haessler et al. [116], Borfeldt [38] and Egeblad [80] that consider this type of restriction.

Relative positioning constraints may require certain subsets of items to be positioned together or a certain distance from one another. Such is the case when a set of items is addressed to the same customer. This placement facilitates cargo loading and unloading operations. Makarem et al. [154], Coello et al. [78] and Terno et al. [207] study these types of constraints.

2.4.1.5 Load-Related Constraints

Load-related constraints have to do with the desirable or necessary properties of the final arrangement of the items in the container.

Stability

In the literature, the stability of the cargo is often considered among the most important properties. An unstable load could cause irreparable damage to the cargo. When dealing with stability constraints, we can distinguish two types of stability: vertical and horizontal.

Vertical, or static, stability prevents items from falling. This means that the base of an item must be fully or partially supported by some element inside the containery [38, 114]. The degree of stability is usually given in terms of a percent. Papers like Araújo et al. [24], Fanslau et al. [91] and Ceschia et al. [50] consider full vertical stability, meaning the load must be 100% supported. Other cases consider the box being supported at least for a pre-specified minimum fraction of the base area, as in Christensen et al. [54], Fuellerer et al. [100] and Tarantilis et al. [206].

Horizontal, or dynamic, stability assures that items cannot shift while the container is being moved. Full horizontal stability means that each item is either adjacent to another item or a container wall [131]. Cases like the one studied by Liu et al. [149] only require one of the surfaces of a small item to be in contact with other small items or a container wall. In practice, the stability of the cargo can be achieved by using additional supports, like packing material (polystyrene), that is placed in gaps between the packed items. This variant was studied by Parreño et al. in [172] and Pisinger et al. in [178].

Complexity

Complexity constraints are related to complex loading patterns. Complex loading patterns may not be feasible for manual container loading since these patterns cannot be visualised in such a way that they can be understood by loading personnel. Complexity constraints reflect these limitations of technological and human resources [29].

2.5 State of the Art

Packing and cutting problems have been studied in various areas of research. The broad range of applications covered by this class of problem has caused this field to expand quickly. A review of the literature reveals that this family of problems has been studied for over half a century [46, 133]. The literature contains many examples of one- and two-dimensional problems, which have been studied more in-depth historically. The study of three-dimensional problems has received relatively less attention. It has not been until the last two decades that three-dimensional problems have generated more interest in the scientific community.

Included in the three-dimensional problems are those related to the loading of containers or pallets. These are particularly relevant in the industrial sector, specifically in terms of loading and transporting cargo. A review of the literature from recent years shows an increasing number of studies focusing on these types of problems, in which a set of small items must be loaded into an object with large, fixed dimensions while maximising the volume loaded.

The increase in the number of publications involving cutting and packing problems created the need to conduct reviews of the literature to allow the scientific community to learn of existing studies having to do with the various types of problems. This has led to the publication of several bibliographies that attempt to group the problems into categories and solution methods [76, 77, 204]. Recently, and due to the impact that the study of three-dimensional problems has had, bibliographic reports have been proposed that focus solely on these types of problems [43, 222]. Most of the bibliographies use considerations similar to those proposed by Wäscher et al. in their typology [215]. In other words, they consider the main properties like dimensionality, the variety of large objects, the variety of small items, the shape of the items and the objective.

Of note in the publications involving three-dimensional cutting and packing problems is the enormous presence of problems like the SLOPP and the SKP. They are also the most widely covered in terms of the three-dimensional problems. Other problems that have also been studied, but more incidentally and with fewer publications, include the SBSBPP y ODP/S, while problems like the MILOPP, MIKP, MHLOPP, MHKP and RCSP have been largely ignored.

There is an increasing number of studies that apply relevant physical constraints to be considered when placing items inside larger objects, such as containers, vans or trucks. Applying restrictions means solving problems that approach real-world conditions, making them more applicable in today's world. Despite this, there are few papers in the literature that consider several restrictions simultaneously.

Part II

Three-Dimensional Packing Problem

Chapter

3 Solving the Multi-Objective 3D Packing Problem

In this chapter we consider the Three-Dimensional Packing Problem (3DPP). This problem arises in actual logistics industries. This class of problem belongs to the well-known Three-Dimensional Packing Problems, which are a generalisation of the one-dimensional and two-dimensional problems. While cutting problems focus on the best possible use of materials, packing problems focus on the best possible use of space. Specifically, this chapter focuses on studying the techniques for solving one of the problem formulations, namely the loading of a single container of fixed dimensions while maximising the volume used. To this end, we conduct a bibliographic review of the problem before giving the mathematical formulation for the problem to be considered in the chapter. Due to the very large search space posed by this problem, the application of exact methods is not viable. As a result, the most common methods applied in the literature to solve the single-objective formulation of this problem are heuristics and metaheuristics. Thus, in the rest of the chapter we will refer to the use of MOEAs to solve the problem proposed.

3.1 Introduction

Container transport is the most widespread logistical commercialisation method in use internationally. The use of containers to store and transport cargo began in the early 19th century [27]. The rapid growth of the technology has led to large improvements in industrial processes. In this context, the container transport industry has been the focus of important improvements that have had positive effects on the quality of some of the processes carried out in this industry, like the loading and unloading of cargo. And yet, the revenue generated by the loading and unloading operations have undeniably been a key factor for the implementation of actions intended to develop this industry. The efficient transport of cargo has a high financial impact on logistics, such that if cargo is loaded in an optimal manner, costs are reduced throughout the entire distribution system [126]. These financial benefits have had a significant impact in the area of computer research in an effort to provide improvements in the search for solutions. Considering these aspects, it is normal for companies that engage in this activity to find themselves in a highly competitive environment. There is thus no doubt that the financial values of these companies depend on the efficiency of the loading and transport of cargo. In logistics, using the available resources efficiently is fundamental. Associated with this is offering quality services while attempting to ensure the good condition of the cargo. It is in this context that the Three-Dimensional Packing Problem (3DPP) or Container Loading Problem (CLP) arises as a field of research. Due to its close ties to the real world and the diversity of its applications, this problem can have many variants, such as the possibility of using one or several containers, containers with fixed or open dimensions, and other options. A study of these computational problems requires clearly defining the objective to be achieved, such as maximising the cargo inside the container. However, most of the real problems that affect the industry present many objectives to be optimised simultaneously.

3.2 Problem Formulation

The most commonly cited applications for the Three-Dimensional Packing Problem (3DPP) are container loading or truck loading in the transport sector and distribution industries. It is a relevant problem in the general classification of cutting and packing problems referred to in the literature as three-dimensional cutting and packing or container loading. Among the cutting and packing problems, container loading problems have been highly studied [222]. In general, when solving the 3DPP, the objective is to arrange a set of three-dimensional rectangular pieces (boxes) inside one large rectangular object (container) so as to maximise the total volume of packed boxes, i.e. trying to obtain a pattern of packaging that uses the container space as much as possible.

The loading process is considered one of the most important tasks in supply chains, which is commonly encountered in the transport sector and wholesaling industries. Efficiently loading a container has both financial and environmental implications. The constant increase in fuel prices makes the loading and unloading processes take on increasing importance among companies involved in container loading or truck loading. Thus, maximising the space used in containers translates into minimising the number of trips that a transporter must make, and in turn minimising the number of containers or trucks needed to transport or store the cargo [173]. The efficient loading of containers not only has financial benefits. Minimising the empty space in containers also addresses the ecological problem of increased traffic and its repercussions on environmental resources [214]. It thus comes as no surprise that this problem has often been considered in operations research literature.

When solving the real 3DPP, normally the most important goal is to find a packaging pattern that maximises the space used. However, in many real-world situations there are many other issues that may be taken into account. In [31] many practical requirements, such as restrictions on orientation, load distribution, stability, priority boarding or the maximum weight of containers are described. Sometimes, packing every box into a container requires considering their orientation, such as when their contents are fragile. Other times, a good load distribution in the container is needed for proper transport, or a specific order for subsequent land shipments is needed. However, a rather common aspect in the scope of this problem is the weight limit of the containers, since they normally can not exceed a certain weight for their transportation, and they should make the most without exceeding that limit. The rented trucks used to transport the shipment are charged based on the total weight they can transport, regardless of the total volume. Thus, the decision maker prefers to load and ship cargo with a high total weight, rather than one with low total weight. For that reason, in this work we take the total weight as a second objective, which simultaneously tries to maximise both objectives: the volume used and the weight. We are interested in finding a solution for these problems.

This way, the problem can be stated as a Multi-Objective Optimisation Problem (MOP). This formulation of the 3DPP problem focuses on the problem involving placing a given number of rectangular boxes of known dimensions into a single container of known dimensions so as to maximise the container space utilised and the total weight of the boxes. However, these two objectives are not the same due to the weight of the goods, the batch size, and other factors [223]. The multiple objectives typically conflict with one another, i.e., there is no single solution to optimise every objective simultaneously (Figure 3.1); rather, they must be simultaneously optimised. As mentioned in Chapter 1, in most real-world optimisation problems the objectives involved tend to be in conflict with one another, meaning there is no single solution that simultaneously optimises every objective. Instead of a single optimal solution, there is a set of alternative solutions from which the person tasked with making a decision must select the best compromise. For the Multi-Objective Three-Dimensional Packing Problem analysed, often times the size of the items or boxes to be packed in the container is not proportional to their weight. One might think a priori that maximising the total volume implies maximising the weight. But a box can be large and its contents lighter than those of a smaller box. This can happen, for example, in an office supply company, where a paper weight

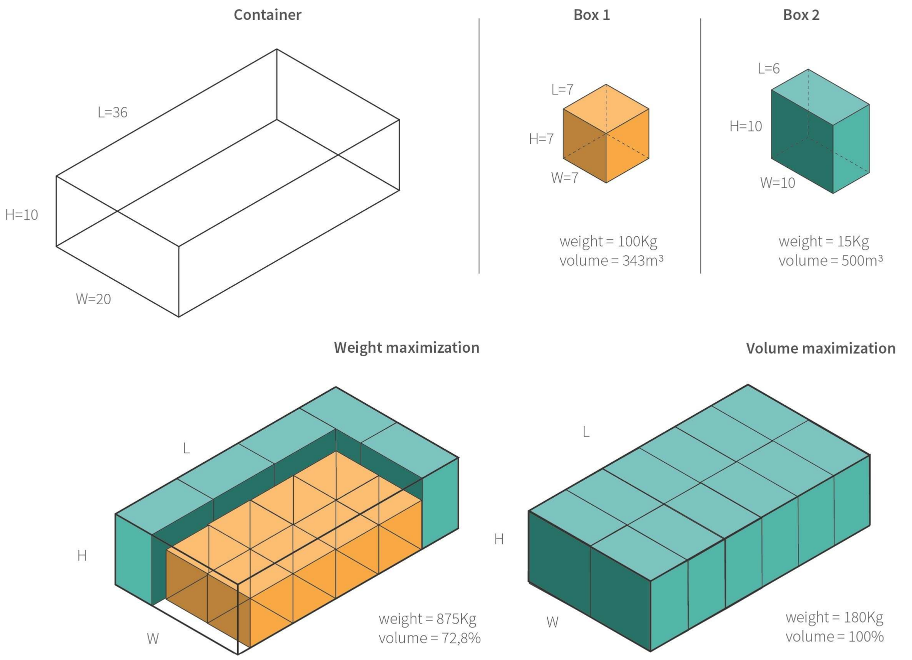


Figure 3.1: Example of conflicting objectives

occupies a very small space and yet is heavier than a trash bin, which is larger and weighs less. Another example might arise in the transport of food, where a box of cereal has a larger volume than a tin of preserves, which can weigh more than a box of cereal. Other times, the weights of the objects to be packed are proportional to their volumes, meaning that large boxes have large weights associated with them, and small boxes weigh less, such as when transporting detergents and toothpaste. For this reason, we can state that both goals have at least some degree of conflict.

Mathematical Formulation

The *Simple Three-Dimensional Packing Problem* is defined as the problem of identifying how to arrange items in a single container. It comprises the maximisation of the loading space used while maximising the weight inside the container without exceeding its weight limits. Thus, it is an intrinsically multi-objective problem.

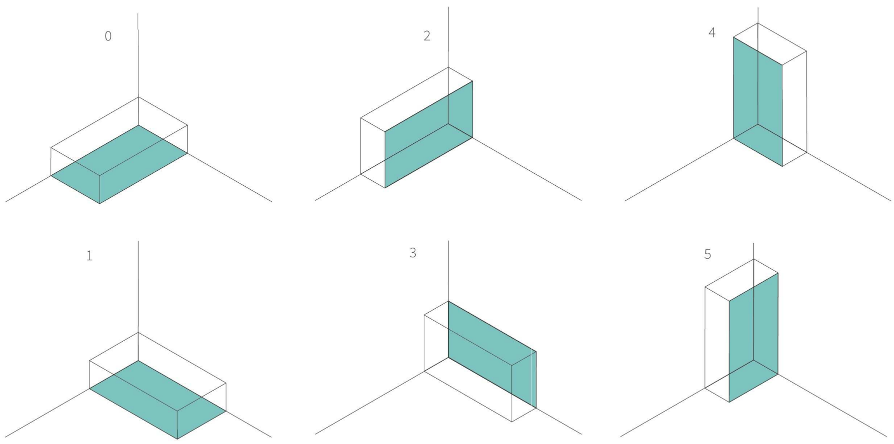


Figure 3.2: Orientations allowed

In this multi-objective approach, only one container is used, with a known width W, length L, height H, maximum weight P_{max} and a set of N rectangular boxes. These boxes belong to one of the sets of box types $\mathcal{T} = \{t_1 \ldots t_m\}$, where the i-th type t_i is of dimensions $w_i \leq W$, $l_i \leq L$ and $h_i \leq H$. Thus, rectangular boxes type can range from weakly homogeneous to strongly heterogeneous. If the set of boxes is considered weakly homogeneous, it means that the instance has a small variety of boxes; whereas a set of boxes is strongly heterogeneous if the instance has more types of different boxes. The boxes can only be placed orthogonally in the container. Associated with each type t_i there exists a weight $p_i < P_{max}$, a volume v_i, a demand b_i, and a number of orientations allowed $o_i \in [0, 5]$. Thus, to indicate the orientation we use a number that ranges from 0 to 5 (Figure 3.2), which indicates how the item is rotated with respect to its original orientation ($o_i = 0$). Each item has two, four or six possible orientation. Normally, there will always be at least two possible orientations for any box (0 and 1), which does not shift its contents since it is a question of rotating the base 90 degrees. In addition, for the known instances of this problem [30, 152], orientations 4 and 5, in which the front side of the box is used as the base, are not allowed if 2 and 3 are not allowed, where the sides of the box are used as the base. Orientations 2 and 3 can be allowed and not 4 and 5. These restrictions stem from the materials contained in the boxes, since they can hold a liquid, for example, which would pose problems in certain positions. Ultimately, the

Property Name	Property Value
Dimensionality	Three-dimensional (3D)
Shape of figures	Regular and with possible orientations
Assortment of small items	Identical or homogeneous to heterogeneous assortment
Assortment of large items	One large object
Type of assignment	Output (value) maximisation
Objectives	Maximise the total available volume and weight for the large object

Table 3.1: Properties of the 3DPP studied

aim is to find a packing arrangement in the container without overlapping, with x_i boxes of type t_i maximising the total volume and weight packed:

$$\begin{gathered}\max \sum_{i=1}^{m} x_i v_i \quad \text{and} \quad \max \sum_{i=1}^{m} x_i p_i \\ \text{subject to } x_i \in [0, b_i] \quad \text{and} \quad \sum_{i=1}^{m} x_i p_i \leq P_{max} \quad \text{and} \quad \sum_{i=1}^{m} x_i v_i \leq W \times L \times H\end{gathered} \tag{3.1}$$

Arranging boxes into a container, truck, or pallet is one of the more complex packing problems with respect to real-world constrains [222]. In this work, the problem will be solved using the following assumptions:

- Each box is placed in the container floor or on top of another box.
- The loaded boxes cannot overlap.
- The stability of the distribution of the boxes is not considered, since we assume the use of filler material to prevent potential problems.
- The number of orientations allowed is restricted. There will always be at least two possible orientations for any box (rotation only considered for the base of the box), since that does not affect its contents. Thus, the dimensions of boxes can be interchanged to modify the orientation.
- The boxes are rigid enough to lie in any location.

The main characteristics of the 3DPP considered in this dissertation are summarised in Table 3.1, based on the general classification presented for cutting and packing problems given in Section 2.2. There may be a single container or multiple containers. In our case, we are particularly concerned with the Single Large Object Placement Problem (SLOPP), where there is a single container and weakly heterogeneous boxes, the Single Knapsack Problem (SKP) if the set of boxes is strongly heterogeneous and the Identical Item Packing Problem (IIPP). Nevertheless, our problem definition is a variant of the SLOPP, because we consider both homogeneous boxes and heterogeneous boxes.

3.3 Related Work

The Three-Dimensional Packing Problem (3DPP) has been widely studied. In the literature, isolated exact algorithms have been proposed to solve it. These studies are scarce, possibly due to the difficulty in representing possible patterns or practical constraints. In [120], the authors present two exact algorithms that are an adaptation of Gilmore and Gomory's approach [112]. These approaches seem suited to solving some classes of non-standard packing problems that consider some restrictions such as fixed positions or orientations of specific items, static balancing and non-rectangular domains. There are papers that solve the container loading problem using Mixed Integer Linear Programming (MIP) models, where the aim is to minimise the unused volume of the container [52, 92]. A basic mixed integer programming model to treat an orthogonal three-dimensional packing problem with rotations is presented in [93]. Junqueira et al. [130] propose an approach based on a MIP model, including multidrop constraints, where they consider both vertical and horizontal stability constraints and the load-bearing strength of the small items.

Most results of this scope are obtained using heuristics and metaheuristics, because, computationally, the 3DPP is an NP-hard problem [188], meaning an exact solution cannot be obtained in polynomial time. Moreover, these problems are very challenging in practice. Although there are many heuristic algorithms [31, 107, 109, 118, 178, 181, 188, 214, 225] in recent years the attention has shifted to metaheuristics such as genetic algorithms [87, 105, 117, 220], simulated annealing [47, 90, 129], tabu searches [35, 148, 172], greedy randomized adaptive search procedures [164], variable neighbourhood searches [175], tree searches [41, 150], and hybrid algorithms [38, 71, 149, 164]. Sometimes, these processes may include parallelisations. Such is the case of some parallel genetic algorithms [106, 113], parallel tabu searches [40], or parallel hybrid local searches [153].

Most approaches deal with single-objective formulations of the mixed integer linear programming, in which the most common objective is to maximise volume utilisation or to minimise the unused volume inside the container. Unlike the large number of approaches proposed for the single-objective formulation, the multi-objective formulation has been studied less. As a result, the references dealing with a multi-objective formulation of the problem are almost non-existent [43]. In fact, one of the best-known works in the literature that addresses the multi-objective problem studied herein uses a simulated annealing algorithm hybridised with filling heuristics [70]. In [70] the authors consider two objectives, the same as in this work: the maximisation of the weight of the boxes loaded in the container and the maximisation of the total volume occupied inside the container. The method proposed in [70] applies different weights to the two objectives considered, yielding - with each execution - a unique and different solution. To achieve a complete set of candidate solutions, the algorithm must be executed several times.

Some formulations of the problem consider the possibility of having multiple containers. As in the case of the previous formulation, in the literature there are only a few exact algorithms [157, 218], many heuristic algorithms [19, 50, 63, 84, 124, 159, 172, 199, 224] and some hybrid algorithms [147]. In other cases, one dimension of the container can be infinite or open, so that all the pieces must be put inside of it [18, 42, 218]. In this work we do not focus on this type of formulation because normally companies have a set of containers or trucks that are not of arbitrary length; rather, they have specific and unchanging dimensions. On the other hand, we can find formulations of the problem in the literature that take into account pieces with irregular shapes [53, 78, 217]. However, most of the items to be transported are packed into rectangular boxes, and not into irregular packages.

In the literature we can find a wide range of packing heuristics and metaheuristics that can be developed by taking into account different specific distributions of packing boxes inside the container. These methods have a direct influence on industrial cargo loading and unloading processes, and help to decide how to put the boxes into the container. Pisinger [178] proposed a classification based on four possible approaches: wall-building, stack-building, cuboid-arrangement and guillotine-cutting. However, Fanslau and Bortfeldt [91] classified them as wall-building, stack-building, horizontal layer-building, block-building, and guillotine-cutting. These approaches are commonly known as the construction heuristic. Based on this, we present some possible alternatives [91, 222]:

- Wall-building approach, which fills the container with a number of walls (vertical cuboid layers) across the depth of the container (Figure 3.3). This procedure is used in [31, 32, 107, 109, 119, 164, 178].
- Stack-building approach, which packs the boxes into stacks or towers that have a base box directly placed on the floor of the container in a way that saves the most space. The stacks do not form walls as defined before. Some authors have used this approach in works like [105, 111, 124].

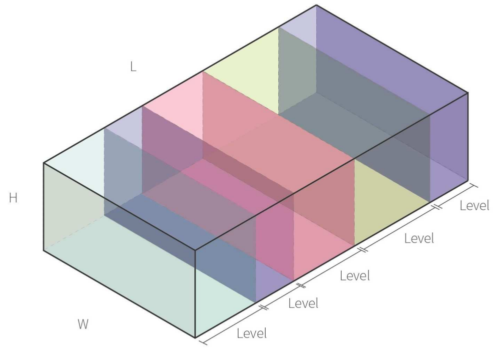


Figure 3.3: Wall-building approach

- Cuboid arrangement approach, which recursively fills the container with cuboid arrangements. A cuboid layout provides sufficient support for the boxes. This approach is presented among others in [35, 36, 84].
- Horizontal layer-building approach, in which the container is packed in horizontal layers from bottom to top, attempting to cover the largest possible part of the load surface of the container. This approach is used in the methods in [29, 207].
- Block-building approach, which fills the container with cuboid blocks. Blocks are homogeneous and each block consists of only identical boxes, although some works consider blocks formed by boxes of different types. [149, 181, 221] describe this approach.
- Guillotine cuts approach, which is based on a slicing tree representation of a packing plan. Each slicing tree corresponds to a successive segmentation of the container into smaller pieces by means of guillotine cuts, whereby the leaves correspond to the boxes to be packed. This approach was studied in [72, 163].

When loading a container with a weakly heterogeneous mix of boxes, the most common approaches are wall-building and horizontal layer-building. In general, construction heuristics select boxes or groups of boxes to place them in a given location inside the container. These processes are combined with a search for a potential position in which to place each box into the container. The definition of these methods have a direct influence on the problem's general solution. Some of the most widely used in the literature are the following [222]:

- Lai and Chan [139] propose the maximal-space strategy. Maximal-space intervals are the set of largest cuboid spaces that entirely cover the unpacked volume of the container. The boxes are packed in the closest space to the bottom left corner of the container that is large enough to accommodate the selected box in each iteration. Gonçalves and Resende [114] present a more efficient method based on this. They identify four parts: identified maximal spaces, prioritised maximal space, creating layers of boxes and joining maximal spaces. The main idea is to identify maximal spaces and then fill them with the best configuration of identical boxes based on a criterion. In each iteration the maximal space with the minimum distance to a corner of the container is filled, with space volume used to break ties.
- Ngoi et al. [166] avoid the dependency of the layout on the order of the boxes by evaluating every potential placement location for each box. In order to improve the efficiency of their approach, they use a spatial representation method. A single 3D matrix (or the spatial matrix), in the form of the layers of 2D matrices, is used to represent the positions and dimensions of all the packed boxes and empty spaces. Other approaches were studied in [28].
- Martello et al. [157] define corner points as the non-dominated locations where an item can be placed into an existing configuration. If we consider the packing profile over the 3D surface, these points arise from contact between the face of two boxes or a box and the container and are located at the point where three edges coincide to create a fully concave corner. Some improvements were described in [58], where the idea provides the means to exploit the free space defined inside a packing configuration by using the shapes of the boxes already inside the container.
- Caving degree approach, defined by Huang and He [124, 125] as packing boxes into a corner or even a cave, an empty hollow-like space surrounded by many boxes, whenever possible. The caving degree approach stores all available corners into a list and evaluates all combination of corners, types of boxes, and allowed orientations, selecting the best combination of corner, box, and orientation based on racking rules. Modifications based on this approach are given in [118].

3.4 Multi-Objective Approach

Applying an exact method to solve the multi-objective 3DPP is not viable due to its magnitude and complexity. As a result, we have opted to develop an approximate proposal using a method that approaches it directly as a multi-objective problem.

Many algorithms have been proposed for solving the 2D/3D Cutting Stock Problems, and most of them concern single objective optimisation rather than multi-objective optimisation problems. We can find a wide range of heuristics in the literature that simplify the 3DPP by converting it into a single-objective problem. Since exact approaches are practically infeasible for most MOPs, a wide variety of (meta)-heuristic algorithms have been designed. A common approach for simplifying the solution to MOPs are the decomposition methods. Such approaches employ a scalarising function to aggregate all the objectives into a single scalar objective function. This way, from the application of a scalarising function to weighting the different objectives, the multi-objective problem is converted into a certain single-objective problem. To obtain different Pareto optimal points, a set of weighting vectors can be used which would result in a set of single-objective subproblems [110].

The Weighted Sum Method is the simplest decomposition approach and probably the most widely used classical method. This method scalarises the set of objectives into a single objective by multiplying each objective with a user supplied weight. The values of the defined weights will depend on the relative importance that the user gives to each objective. In any case, the set of defined weighting vectors will restrict the set of single-objective solutions to be obtained. The advantage of these methods lies in the fact that it is straightforward to apply standard single-objective methods to solve the resulting single-objective subproblems. Another method used to convert the original problem into a single-objective one is the method with restrictions, which relies on transforming some of the objectives into restrictions by employing Pareto optimals.

Optimising a combination of the objectives has the advantage of producing a single compromise solution - which can be achieved by applying classical single-objective optimisation strategies - requiring no further interaction with the decision maker. However, the application of such an approach requires prior knowledge of the problem, which is not always available. Moreover, these techniques lose in solution diversity and may be sensitive to the shape of the Pareto-optimal front, e.g. if the optimal solution cannot be accepted, either because the function used excludes aspects of the problem that were unknown prior to optimisation, or due to an inappropriate setting of the coefficients of the combining function, new runs of the optimiser may be required before a suitable solution is found [226]. Therefore, a MOP solution calls for alternative approaches. Usually, a more appropriate approximation involves the application of techniques that can specifically deal with multiple objectives and the intrinsic complexities of MOPs (very large search spaces, uncertainty, noise, disjoint Pareto curves, etc).

3.4.1 Representation of Candidate Solutions

The MOEAs have shown promise for solving problems in the area of cutting and packing. For this reason, we have decided to apply MOEAs to solve the 3DPP. When a MOEA is designed to solve a specific problem, it has to take into account several decisions involving its design. It is necessary to design a scheme representation for the candidate solutions, a set of evolutionary operators and methods for the evaluation of the candidate solutions. These decisions are crucial and are related to the solution method that is eventually adopted. One of these decisions concerns the representation of the solutions, which must contain information about the solution it represents. A mechanism is thus needed to code the individuals within the population. The choice of which coding method to use will depend to a large extent on the problem to be solved, since it might be that one coding is optimal to solve one case, but that same coding complicates the solution in another case. As a result, it is important that the coding scheme allow for an adequate representation of the solutions, and that the coding be optimal for the problem at hand. According to [39], different groups of metaheuristic proposals can be identified for cutting and packing problems depending on the type of coding selected to represent the solutions:

- The methods of the first group use a coding of solutions. Typically, an encoded solution stipulates a placement sequence for the pieces. The metaheuristic search is carried out in the space of the encoded solutions and normally uses problem-independent operators. A placement or decoding routine transforms encoded solutions into complete layouts.
- Solution approaches for the second group have an intermediate position. While on the one hand encoded solutions already contain, to a certain extent, layout or geometrical information, an additional placement routine is also required for the final positioning. Typical for this group is a problem-specific coding of solutions, which is often based on graphs, and the use of corresponding problem-specific operators.
- The approaches of the third group do not use coding. The search is carried out directly in the space of fully defined layouts, which are therefore manipulated as such by specific operators.

For the problem at hand, we have opted to use the coding in the second group, indicating the order and orientation in which each piece must be placed inside the container. This requires a construction heuristic that allows us to ascertain the final location of the pieces. We have opted not to use a direct representation of the individuals through postfix notation, since obtaining feasible solutions in an acceptable time frame is not viable.

		G_1	G_2	G_3		G_{s-2}	G_{s-1}	G_s
piece type	t_i	t2	t4	t1	...	t2	t3	t1
number of pieces	n_i	4	1	2	...	2	2	3
pieces orientation	r_i	1	0	1	...	4	0	2

Figure 3.4: Scheme to represent a candidate solution

Thus, in order to represent candidate solutions, we have defined the structure shown in Figure 3.4. If $s \in [1, N]$ is the chromosome size - which varies according to the grouping of pieces - an individual will consist of a sequence of genes $G_1, \ldots, G_s$ where each gene consists of three elements (t_i, n_i, r_i), with $t_i \in \{t_1, \ldots, t_m\}$, $n_i \in [1, b_i]$ and $r_i \in [0, o_i)$. Thus, it is a sequence formed by the piece type (t_i), the number of pieces of that type (n_i) and the rotation for those pieces (r_i). This will determine the order and orientation in which the pieces will be put inside the container. A valid chromosome must contain all the pieces of each different type. Using this representation, the chromosome size may vary, depending on how the pieces of the same type are grouped:

$$\forall j \in [1, m] \ \sum_{k=1}^{s} n_{jk} = b_j \ \begin{Bmatrix} n_{jk} = n_k & \text{if} & t_k = t_j \\ n_{jk} = 0 & \text{if} & t_k \neq t_j \end{Bmatrix} \tag{3.2}$$

We assume that every box loaded can be placed anywhere inside the container, and that they can change their orientation. Each piece has two possible orientations allowed, as determined by the input instance. That is, the dimensions of the base are interchangeable. Thus, to indicate the orientation r_i of a set of pieces we use a digit from 0 to 1, indicating a rotation of the piece with respect to its original orientation. This assumption can be relaxed in the 3DPP definition and we can consider two, four or six possible orientations (Figure 3.2).

3.4.2 Generation of the Initial Population

Every evolutionary algorithm starts from an initial population. This initial population can be determined at random, by applying some method to initialise it or based on known solutions from prior experience. However, generating the initial population at random ensures that the search process begins with a set of sufficiently disperse and varying solutions. Based on this initial population, or initial set of solution, the natural evolution process commences.

For the initial generation of individuals for the loading problem, first a type of piece t_i must be randomly generated, i.e. there are still pieces of that type to be placed, meaning that the number of pieces of type t_i placed inside the container, x_i, is smaller than the total number of pieces of that type available, b_i ($x_i < b_i$). The amount of pieces of that type, n_i, is then generated, this being the number of pieces of that type to be considered in the current gene. This amount must be smaller than or equal to the number of pieces of that type remaining to be placed, $n_i \leq b_i - x_i$. Finally, the rotations allowed for this set of pieces $r_i \in [0, o_i)$ are also randomly generated. These steps must be carried out until all the pieces have been entered into the chromosome, i.e., $\forall i \in [1, m]$, $x_i = b_i$.

3.4.3 Evaluation of the Objectives

Every evolutionary algorithm consists of an evaluation function that determines if the individuals that comprise the population offer good or bad solutions for the problem at hand. This is the most costly phase of the evolutionary model for a real application. Evaluating a solution means calculating the value of the objective function. For the problem at hand, each chromosome represents a certain order - and orientation - in which pieces are considered to be introduced in the container. The coding employed provides the information needed to determine whether a piece fits in the container or not, but a placement or filling heuristic still has to be applied to decide exactly where the pieces should be located. Based on this information, we know exactly which pieces have been loaded into the container, and both optimisation objectives considered - total volume and weight - can then be evaluated.

The placement or filling heuristics proposed to evaluate the possible solutions to the multi-objective optimisation problem defined in Section 3.2 are based on construction approaches such as the wall-building approach [91] and the vertical layer-building approach, which is a variation of horizontal layer-building [91]. When working with the wall-building approach, walls or vertical cuboid layers are created along the container, that is, parallel cuts are made going from the rear to the front of the container. In contrast, when working with the vertical layer-building approach, the cuts are perpendicular to the rear of the container, meaning cuts are made that divide the depth of the container into vertical layers. The set of heuristics proposed are problem dependent and take into account the conditions and restrictions given in Equation 3.1. Thus, extrapolating them to another problem would be complex. All heuristics are based on the creation and management of piece layers or walls within the container. These layers or walls identify empty spaces inside the container, and thus represent areas in which to place items.

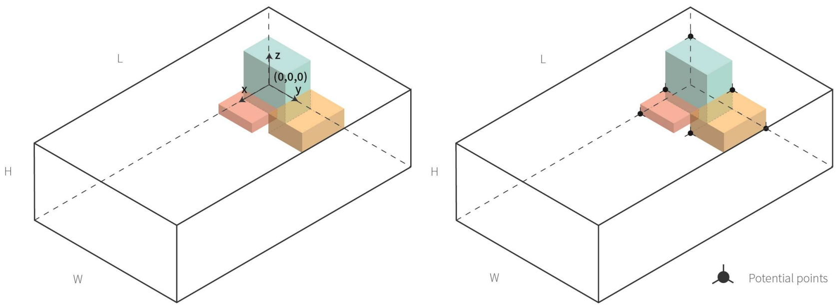


Figure 3.5: Potential corners

To represent the loading patterns, we use the Euclidean plane $\Re^3$ with a Cartesian coordinate system (x, y, z), where the x-axis represents the length of the container, the y-axis its width and the z-axis the height of the container. This same scheme is used to represent the location of the items that are packed into the container, and it can be used to determine the arrangement of loaded boxes inside the container. These heuristics are based on the selection and subsequent placement of pieces inside the container. To select the exact location of the pieces, we used an adaptation of the corner points strategy [157], which identifies all of the potential points inside the container; that is, points at which the pieces can be positioned. These points coincide where the pieces intersect one of the faces of the container, or where the pieces intersect with one another. Thus, all of the back-left-bottom corners that fulfil the previous condition are potential corners (Figure 3.5). The point $(0, 0, 0)$ in Cartesian coordinates is considered the origin and the first position to be occupied inside the container.

In this way, we have defined four different heuristics for decoding the individuals. So as to simplify their nomenclature and standardise the terminologies, we use the term level to encompass the wall and layer concepts presented earlier. A level is the guillotine cuts that cross the container walls from one side to the other and that are used to subdivide the interior volume of the container into smaller cuboids. The levels created inside the container are rigid, meaning it is not possible to locate a piece between different levels.

All of the heuristics presented in this section begin from an initial chromosome, presented in Section 3.4.2, which determines the sequence for selecting the pieces $q_1, \ldots q_N$ in their respective orientations. The pseudo-code for the four heuristics

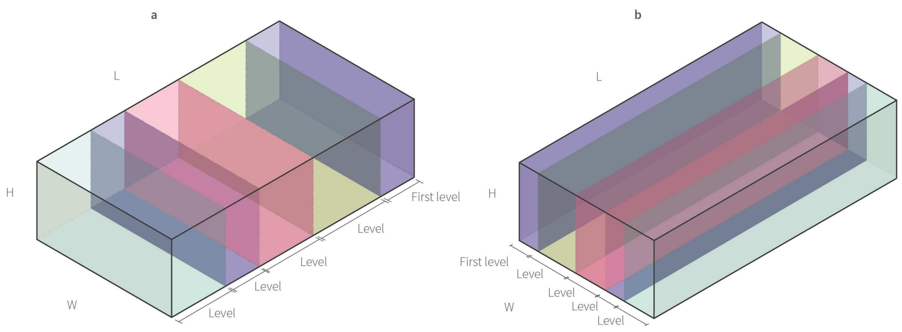


Figure 3.6: Parallel guillotine cuts: a) x-axis cuts b) y-axis cuts

is included in Appendix B. Based on this information, each heuristic works as described below.

3.4.3.1 Single-Level Filling Heuristic in Length

The Level Filling Heuristic in Length is based on storing pieces inside the container using levels, created by guillotine cuts parallel to the x-axis (Figure 3.6-a).

In the case of the Single-Level Filling Heuristic in Length (SLFHL), each level is independent, that is, pieces are placed in level L_i until it is impossible to place any more pieces in that level. When this happens, a new level L_{i+1} is created adjacent to the current level, leaving the level L_i not available to allocate more pieces. Thus, SLFHL never uses the next layer if there are other empty spaces in the current level to which items can be allocated. New levels will continue to be generated until the upper positive limit of the y-axis is reached, there are no more items to be packed or the free space in the container along the y-axis is not sufficient to contain the remaining items.

As noted earlier, the heuristic starts from a given sequence of genes generated at random. Each gen has an item type t_i associated with it, with its corresponding weight p_i and specific dimensions (w_i, l_i, h_i), where w_i represents the width, l_i the length and h_i the height of the item type based on its orientation o_i. In the initial iteration, and based on this information, the i-th gene of chromosome $G_i = (t_i, n_i, r_i)$ is chosen such that its w_i fits in the width W available in the container and its placement does not cause the container's weight limit P_{max} to be exceeded. At first, the only possible position is the back-left-bottom corner ($(0, 0, 0)$ in the Cartesian

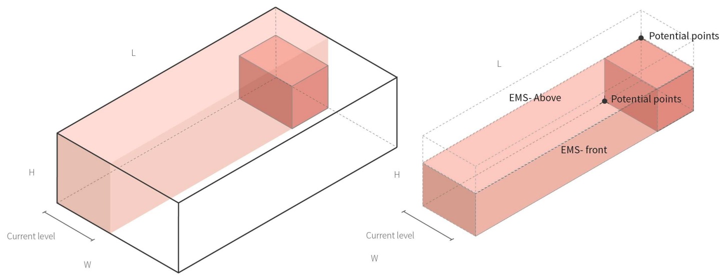


Figure 3.7: Creation of initial EMSs for the SLFHL

coordinate system). Thus, the first item from the n_i that make up the G_i is selected and packed inside the container. It is at this instant that the first level is created inside the container. Associated with each level are specific dimensions. In the case of the SLFHL, every level is the same length and height, both of which correspond to the container's length L and height H. The width, however, is determined by the width w_i of the first box loaded in each level (Figure 3.7). The remaining levels are parallel guillotine cuts to the previous cut. Each item can only be contained inside one level.

In order to determine whether a piece can be locate inside a container or not, the heuristic keeps track of a sorted list of Empty Maximum Spaces (EMS). When a piece is located inside the container, this one generates different possible holes (EMSs). Each hole is stored in different lists, with its volume, dimensions and the Cartesian coordinates (x,y,z). These lists are sorted by the volume of EMS, from smallest to the largest. The Cartesian coordinates associated with each EMS correspond to the potential points, that is, the back-left-bottom points of each EMS (Figure 3.7). As a result, the SLFHL works as follows:

- The first item of the chromosome that fulfils the dimensionality ($w_i < W$) and weight ($p_i \leq P_{max}$) conditions is packed inside the container. This item determines the width of the level, as well as the dimensions and potential points of each EMS. When the first item in the level is packed, two EMS are generated, one in front of the box (x-axis - positive direction) and another above the item (z-axis - positive direction). Each of the EMS is defined as an independent list, EMS-front list and EMS-above list. An example is shown in Figure 3.7.

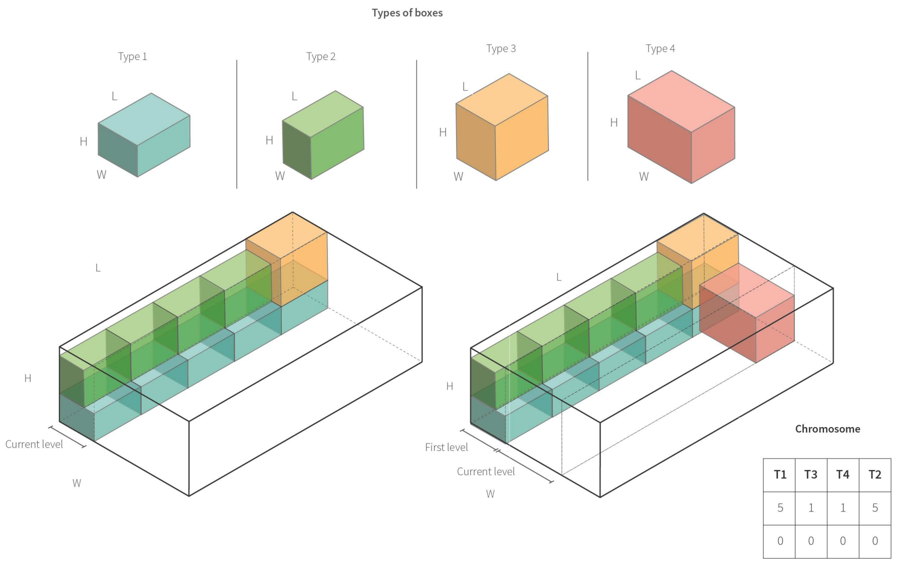


Figure 3.8: SLFHL example

- The next item in the sequence of items determined by the chromosome is selected, observing the weight limit P_{max} of the container. If the item can be packed inside the container, the necessary EMS are generated, creating a maximum of three EMS, one in front, another above and a last one alongside the packed box. If this item cannot be packed into the current level, all other items of the same type are ruled out and an attempt is made to pack the next item of a different type in the sequence. In other words, no more items of the same type will be attempted to be placed in the current level ($t_i \neq t_j$), where t_i is the type of item already evaluated and t_j is the new item type to evaluate. Such is the case of item type 4 of the chromosome shown in Figure 3.8, which cannot be packed into the first level and which will be attempted in future levels. When the entire sequence of items has been attempted and none can be packed into the current level, a new level adjacent to the previous one will be created along the positive direction of the y-axis, and all of the EMS created for the current level will be deleted. Thus, the EMS-front list, the EMS-above list and the EMS-beside list are freed to hold the new holes created in the new

Algorithm 5 Placement Heuristic

1: First, we try to introduce the item in the EMS-front list with the smallest volume, such that the smallest possible space is wasted. If there is no available space for our current item in the EMS-front list, we check for available space in the EMS-above list. If even then the current item does not fit, we try to place it in the EMS-beside list.
2: When the item fits into a EMS, the box is placed at the back-bottom-left corner of the EMS. In such a case, we must check if the box fits into the EMS without leaving any empty space. If so, we can removed the EMS from the corresponding EMS list. If not, it means that there is still some remaining free space in front, above and/or beside the already positioned box, meaning we must create a set of new - in front, above and/or beside - EMS.
3: (In Single-Level Filling Heuristic): if it is impossible to pack the item in the current level, then it is ruled out for the current level and an attempt will be made to pack it in the next levels that are created.
4: (In Multiple-Level Filling Heuristic): if the item does not fit in any available EMS lists, then a new level is opened.
5: When an item is placed in the container, the EMS lists are sorted.

level. Once the new level is created, the sequence of items will be run through again to attempt to pack those that could not be packed into the container earlier. The width of the new level will be given by the width of the first item packed into it, and its length and height will be those of the container (Figure 3.8). In order to place the items on a level, the placement heuristic presented in Algorithm 5 is used. Note that point 4 in the algorithm is not considered for the SLFHL heuristic.

- When it is not possible to fit items in the available EMSs, the procedure finishes, and no more items are loaded into the container. It is also possible for the heuristic to finish because the container's weight limit P_{max} has been exceeded, at which point all possible items will have been packed inside the container. It is now possible to compute the value of the objectives (total volume and weight) by adding the volume and weight of all the items loaded.

3.4.3.2 Multiple-Level Filling Heuristic in Length

The filling heuristic presented in this section is called Multiple-Level Filling Heuristic in Length (MLFHL), which is a modification of the Single-Level Filling Heuristic in Length (SLFHL) described earlier. Both heuristics are based on selecting items and holes within the container in which to place the items until the preset stopping condition is reached. The construction method used for this new heuristic is the same as that used for the SLFHL, and is based on the generation of levels. The difference between SLFHL and MLFHL is that SLFHL packs items into a single level with each

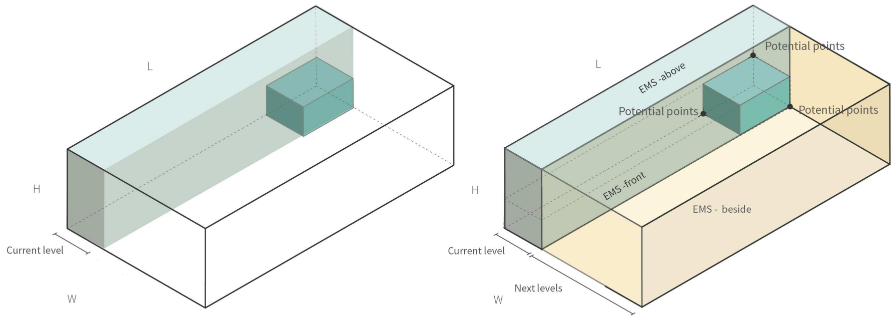


Figure 3.9: Creation of EMSs for MLFHL

iteration. Items cannot be placed in a level different from the one being considered at that instant. However, in MLFHL every level is available to accommodate pieces, i.e., if a given item does not fit into the current Empty Maximum Spaces (EMS), it is placed in the next unused level, thus creating a new set of empty spaces. MLFHL works as follows:

- The first piece in the chromosome is introduced at the back-left-bottom corner of the container. This piece will determine the dimensions of the empty spaces to be generated, one in front of the box, one above it and one beside the placed item (Figure 3.9). Each empty space corresponds to an EMS. These EMS contain lists of holes. Thus, the EMS-front list contains all of the empty spaces that remain in front when an item is packed into one of the existing empty maximum spaces.

- The algorithm will then attempt to insert the remaining items inside the container without exceeding its maximum allowed weight P_{max}. In order to place an item, all of the holes available in the different EMS lists are analysed. If an item can be inserted into an available EMS, then the item is packed inside the container, generating the two or three types of empty spaces possible. For example, if an item of type 3 cannot be inserted into the EMS for the first level, then a new level is created using the EMS-beside list (Figure 3.10). Finally, if it is not possible to create more levels inside the container, the item is dismissed and another item of a different type is selected, that is, $t_i \neq t_j$, where t_i is the type of the current item and t_j that of the new item. Items

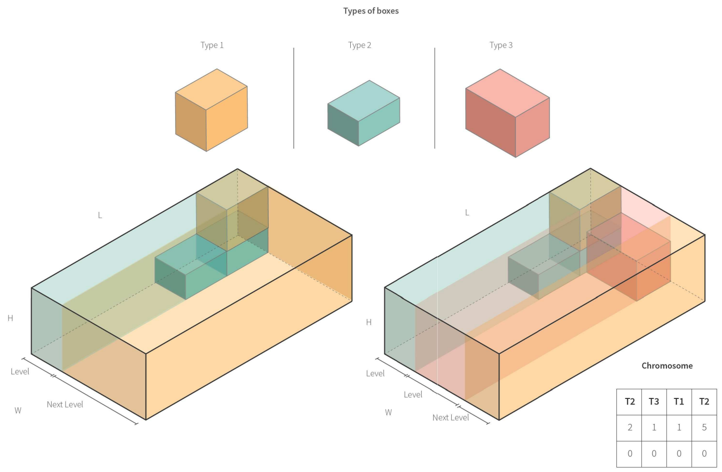


Figure 3.10: MLFHL example

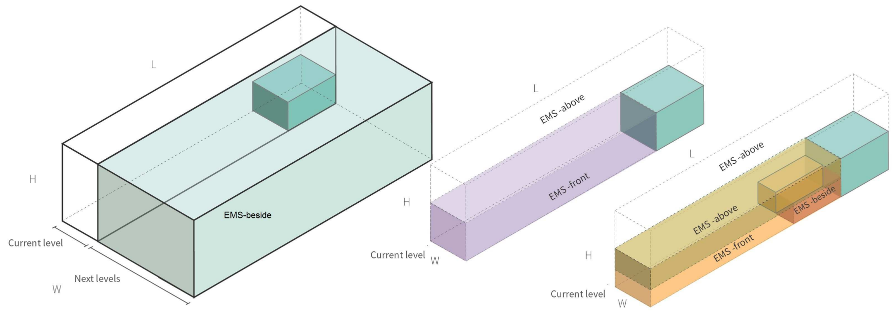


Figure 3.11: Creation of new EMSs

from the chromosome are selected in this manner until the stopping condition is reached. Each time that new empty spaces are generated in the container, the lists of holes are arranged from small to large volumes (Figure 3.11), such that the smallest EMS are attempted to be filled first so as to waste as little space as possible. The method for evaluating the EMS for placing items is shown in Algorithm 5.

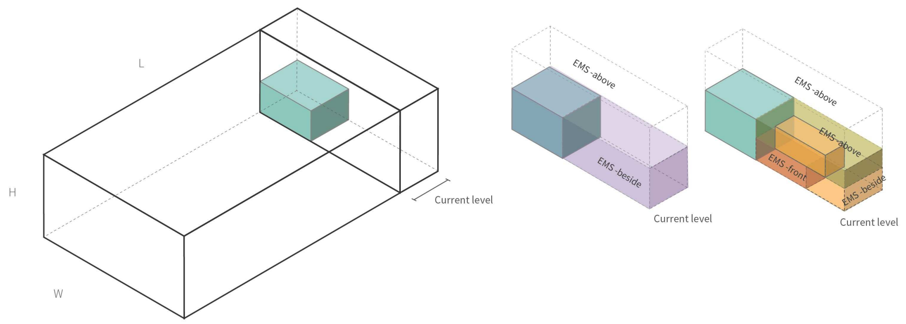


Figure 3.12: Creation of EMSs for SLFHD

- When the stopping condition is reached, the heuristic calculates the value of the objectives, volume and weight.

3.4.3.3 Single-Level Filling Heuristic in Depth

The Level Filling Heuristic in Depth is based on storing items inside the container by levels, with the levels being guillotine cuts parallel to the y-axis (Figure 3.6-b).

Single-Level Filling Heuristic in Depth (SLFHD) relies on the same scheme as SLFHL. The difference between SLFHL and SLFHD is how they create levels or walls. In SLFHL, the levels are parallel guillotine cuts to the x-axis. However, in SLFHD all the cuts are parallel guillotine cuts to the y-axis (Figure 3.6-b), such that now it is the length of the item loaded that determines the width of each level. The W of the container corresponds to the length of the level and the container's height H corresponds to that of the level. The rest of the heuristic functions in the same way as the SLFHL explained earlier (Figure 3.12).

3.4.3.4 Multiple-Level Filling Heuristic in Depth

The filling heuristic we present now is called Multiple-Level Filling Heuristic in Depth (MLFHD). It is a modification of the previous MLFHL. The difference between them is the way in which the levels or walls are created. The walls or levels, in this case, are parallel guillotine cuts across the depth of the container, that is, the guillotine cuts are parallel to the y-axis (Figure 3.6-b). These levels or layers identify empty spaces inside the container, and thus they represent areas where

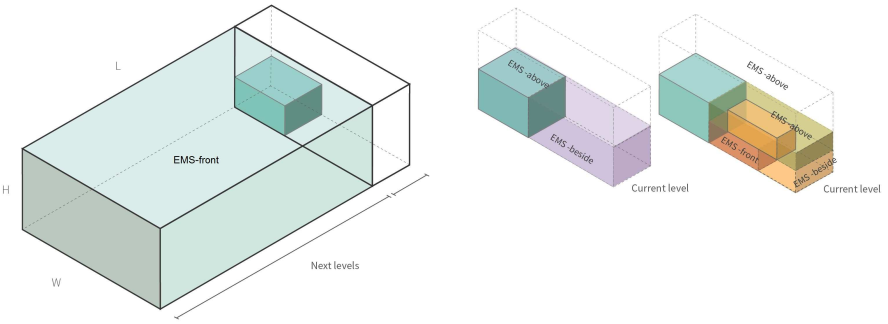


Figure 3.13: Creation of EMSs for MLFHD

items can be placed. As explained with SLFHD, the correspondence in dimensions between the packed item and the level differs with respect to the length heuristics, as shown in Figure 3.13.

3.4.4 Genetic Operators

MOEAs are based on a variation phase in which genetic operators are applied. Each of the genetic operators applied during this phase play a different role in the process of evolving the population and creating new individuals from existing ones. In the variation phase, the so-called mutation and crossover operators are applied to a set of individuals that are selected by each strategy. These operators are capable of introducing diversity into the population, which allows enhancing the individuals that comprise said population. The operation of these operators, however, is not determined by each algorithm; rather, they must be selected independently. The different operator types that are applied must ensure the integrity of the chromosomes; that is, once the operators are applied, each chromosome must still comply with the problem definition presented in Equation 3.2, i.e., all of the items of a certain type must be contained within the chromosome evaluated, and each item must have associated with it an allowed orientation based on the item type to which it belongs. Thus, since we have used an encoding that implicitly represents the problem solutions, the different types of operators must deal with their specific characteristics.

Crossover operator

The crossover operator is intended to combine two or more individuals in the current population, the parents, to yield two new individuals, offspring, with the best characteristics of each parent. In these cases, we speak of a hereditary trait. The crossover operator is subject to a certain crossover probability p_c, meaning there will not always be an exchange of information.

A one-point crossover operator has been designed so that one individual is represented by the chromosome $C1 = (G1_1, \ldots, G1_{s_1})$ and another individual by the chromosome $C2 = (G2_1, \ldots, G2_{s_2})$. This operator works as follows:

- First, it chooses a random gene from each individual, $p_1 \in [1, s_1]$ and $p_2 \in [1, s_2]$.
- It then splits both individuals at this point and creates the two offspring by exchanging the tails, so that, $CO1 = (G1_1, \ldots, G1_{p_1}, G2_{p_2+1}, \ldots, G2_{s_2})$ and $CO2 = (G2_1, \ldots, G2_{p_2}, G1_{p_1+1}, \ldots, G1_{s_1})$.
- The first part of each individual, $G1_1, \ldots, G1_{p_1}$ and $G2_1, \ldots, G2_{p_2}$, is modified with respect to the total number of pieces of each type present in the chromosome. In other words, it checks to ensure that the condition of Equation 3.2 still holds. If it does not, it removes any extra piece types if there are too many pieces of this type, or adds them if they are missing:
  - If we consider how each gene of chromosome $CO1$ consists of a 3-tuple of the form (t_x, n_x, r_x) such that the total number of pieces n_x of a given type t_x exceeds its demand b_x, then pieces of the same type t_x must be removed from chromosome $CO1$. Then, $\forall t_x \in \{t_1, \ldots, t_m\}$ $\sum_{i=1}^{p_1+(s_2-p_2)} n_x > b_x$ the first part of the chromosome $G1_1, \ldots, G1_{p1}$ is analysed to look for the first gene $G1_y = (t_y, n_y, r_y)$ that is of the same type as the piece, that is $t_x = t_y$, and the number of excess pieces is removed $e_x = \sum_{i=1}^{p_1+(s_2-p_2)} n_x - b_x$, such that $n_y = n_y - e_x$, if $n_y > e_x$. In the event that $n_y \leq e_x$, then $e_x = e_x - n_y$ and the complete $G1_y$ gene is erased, shifting the next genes one position to the left. If once these pieces are removed, there are still pieces of type t_x to be removed, $e_x > 0$, the same process is repeated with the remaining genes in the first part of the chromosome, $G1_z = (t_z, n_z, r_z)$ $z \leq p_1$ and $t_x = t_z$, until $e_x = 0$.
  - If each gene in chromosome $CO1$ is a 3-tuple (t_x, n_x, r_x), such that the number of total pieces, n_x, of piece type t_x is less than its demand b_x, then pieces of type t_x must be added to chromosome $CO1$. Thus, $\forall t_x \in \{t_1, \ldots, t_m\}$ $\sum_{i=1}^{p_1+(s_2-p_2)} n_x < b_x$, the number of pieces of type t_x that are missing will be $d_x = b_x - \sum_{i=1}^{p_1+(s_2-p_2)} n_x$. The first part of the chromosome,

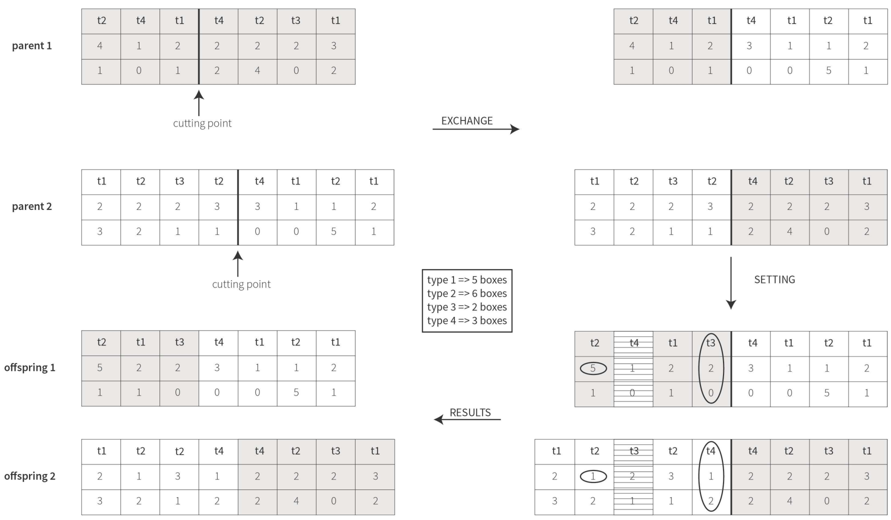


Figure 3.14: Crossover operator

> $G1_1, \ldots, G1_{p_1}$ is analysed to search for the first gene $G1_y = (t_y, n_y, r_y)$ that has the same piece type, $t_y = t_x$, to add it to the parts that are needed $n_y = n_y + d_x$. If there is no gene of type t_x in chromosome $G1_1, \ldots, G1_{p_1}$, a gene will be added to the end of the first part of the chromosome, $G1_{p1+1} = (t_x, d_x, r_x)$ with orientation $r_x \in [0, o_x]$. This will cause the genes in the second part of the chromosome to shift one place to the right.

The same process is carried out on chromosome $CO2$. A general diagram is given in Figure 3.14.

Mutation operator

The mutation process causes some of the genes in the chromosome to randomly change their value. This allows expanding the search into new regions of the space. This operator is not normally used by itself, but in conjunction with the crossover operator. So if the crossover is applied, some of the offspring will mutate with probability p_m. Different operations involving the chromosome have been designed as part of the mutation process. These are presented below:

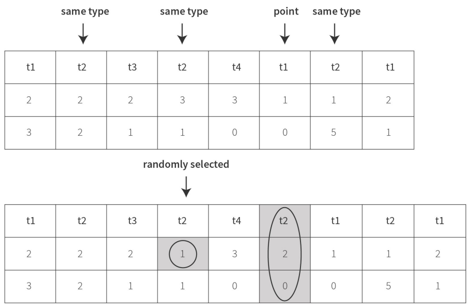


Figure 3.15: Add mutation operator

- *Add one gene*: a type of piece is randomly generated $t_x \in \{t_1, \ldots, t_m\}$. This type t_x has a gene associated with it, G_x; then, the chromosome is searched for every genes of this type of piece. One of these genes $G_y = (t_y, n_y, r_y)$ is then selected at random such that $t_y = t_x$, $x \neq y$ and $n_y > 1$; that is, one with more than one associated piece is selected. A number of pieces $n_x < n_y$ is chosen from the selected gene G_y, so that the pieces are distributed between that gene $(t_y, n_y - n_x, r_y)$ and the new one that will be added (t_x, n_x, r_x). The orientation is chosen from those allowed for that piece type $r_x \in [0, o_x)$. Finally, we choose the position of the chromosome in which to insert the new gene, shifting the rest to the right (Figure 3.15).

- *Remove a gene*: a position x within the chromosome is randomly selected, such that at that position, the gene is $G_x = (t_x, n_x, r_x)$. If the piece type t_x of the selected gene G_x appears more times in that chromosome, then a gene $G_y = (t_y, n_y, r_y)$ is randomly selected from among the same type, i.e. $t_y = t_x$ and $x \neq y$. In order to eliminate the gene G_x, a number of pieces n_x is transferred to gene G_y, such that $n_y = n_y + n_x$. Since it is possible that both genes do not have the same type of orientation for the pieces ($r_x \neq r_y$), one of them is randomly selected. Finally, G_x is eliminated by compaction to the left. In the Figure 3.16 is shown an example.

- *Change a gene*: a random position x on the chromosome is selected, such that that position inside the chromosome corresponds to gene $G_x = (t_x, n_x, r_x)$. Then, the type of orientation r_x for the gene G_x is randomly changed within the possible orientations for the piece type t_x ($r_x \in [0, o_x]$). An example of such a scheme is shown in Figure 3.17

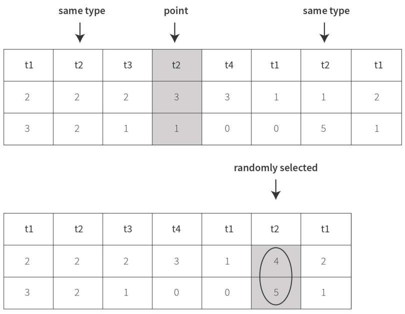


Figure 3.16: Remove mutation operator

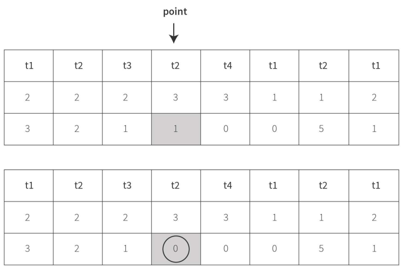


Figure 3.17: Change mutation operator

- *Modify some genes*: The entire set of genes that comprise the chromosome is analysed. Then, for each gene G_x, the orientation type r_x is randomly changed considering the mutation probability p_m; that is, a gene G_x will be modified if it meets the condition $prob < p_m$, where $prob$ is a generated random value. The new orientation r_x is selected from among the orientations allowed for the piece type t_x, excluding the same orientation type r_x (Figure 3.18).

- *Swap two genes*: two genes $G_x = (t_x, n_x, r_x)$ and $G_y = (t_y, n_y, r_y)$ are randomly selected, such that the positions $x \neq y$. Both genes then swap their positions such that gene G_y is placed in the position x, and gene G_x is placed in position y. A diagram of this is shown in Figure 3.19.

- *Shift a gen*: a position x is selected inside the chromosome such that the gene associated with that position is $G_x = (t_x, n_x, r_x)$. A position y is then selected

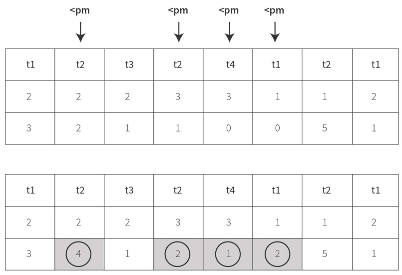


Figure 3.18: Modify mutation operator

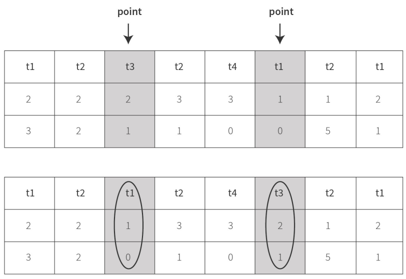


Figure 3.19: Swap mutation operator

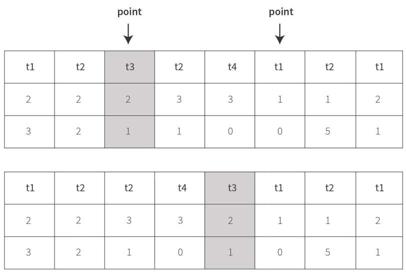


Figure 3.20: Shift mutation operator

at random such that $x \neq y$. Gene G_x is inserted into position y, shifting the remaining genes $(G_y, G_{y+1}, \ldots, G_s)$ to the right. Finally, position x is removed from the chromosome (Figure 3.20).

Chapter

4

Instance Generator

In this chapter we present the design and implementation of an instance generator that addresses the requirements when the multi-objective variant of the Three-Dimensional Packing Problem (3DPP) is considered. In this context, we present a study on the current state of benchmark data, placing special emphasis on the lack of said data when dealing with problems that have more than one objective, and even when the 3DPP is tackled considering some restrictions. We then present the two heuristics developed to generate instances that consider more than one objective. In this chapter we consider an instance generator that takes into account two objectives: maximising the volume and weight. Finally, in an effort to undertake robust computational studies, we generate a set of instances while taking into account the properties of 3D Packing Problems, such as the dimensions of the large rectangular object, different box types, small items of various dimensions, unexpected weight to volume ratios, and others. Some of the problem instances generated are shown at the end of this chapter.

4.1 Introduction

When dealing with a certain formulation of a problem, it is necessary to have a set of problem instances related with the problem at hand, since this can be used to efficiently validate the methods proposed [160]. This set of instances must maintain the problem's properties. For example, when dealing with an output maximisation problem, we must know the number of large objects and their dimensions, whereas for an input minimisation problem, this information is normally obtained as part of the solution. Therefore, each problem type requires a specific set of test data. In practical cases, these data are given by the demands of the users. In many research studies, however, this does not occur, which makes it essential to define a suitable set of data for the problem in question.

When testing solution approaches in cutting and packing research, the problem instances must be generated in some way. It is advisable to employ instances that offer different properties for evaluating a specific problem, since that allows checking the validity of the proposals in an efficient manner. These instances should contain items of different sizes, weights and so on, such that the researchers can determine if the heuristics proposed are suitable for any particular data types. For initial testing, simpler instances may help the designers to easily check the suitability of the approaches and to detect possible errors. However, these small data sets do not offer a determination of whether the algorithms in question will be able to deal with large problem sizes, which are normally better suited to the real demands of cutting and packing problems. At the same time, handling these larger problems as tests for the algorithm can be complex. The computational time required could increase rapidly, which often means that the quality of the solution will be approximate, since it can be very hard to obtain the exact optimal solution.

When an algorithm is proposed to solve a specific cutting and packing problem, one way to determine its quality is to compare it with other approximations used in the literature to solve it. The types of cutting and packing problems and the existence of an associated bibliography make it possible to access other works that attempt to resolve a similar formulation of the problem. Once these works are located, they must be tested using the same set of problem instances. The different proposals can thus be directly compared.

To simplify how the group of problems is selected, we can use the data sets already defined for the works selected. In general, these test instances have been defined by other authors who have worked on the problem. Moreover, for these instances, some approximate or exact solutions are generally already known, and the results generated by other algorithms present in the literature can be consulted. This yields an easy way to check the quality of our proposals. Based on this, we have used several libraries of benchmark cutting and packing problem instances that provide a point of reference:

- DEIS - Operations Research Group Library of Instances [68],
- $PackLib^2$ [171], and
- OR-Library [170].

Euro Special Interest Group on Cutting and Packing (ESICUP) [89] also provides a large set of problem instances. These problem instances are classified by the number of dimensions (1D, 2D and 3D), and by the types of items available (regular or irregular). It is then possible to check the type of assortment of small items and large objects available.

Knowing how the set of instances were generated can also be useful, since this gives the user the possibility of generating new problem instances that are simpler or more complex than those already available. Not all authors give an explicit description of how they generate their data sets, but there are research papers devoted specifically to the generation of test problems [39, 158, 165, 196, 213].

4.2 Existing Benchmark Instances

The literature features a wide variety of problem instances for dealing with cutting and packing problems. However, only some of these data sets concern the 3D Packing Problem. In the specific case of the 3DPP that consider restrictions like those given in Section 2.4.1, which must be taken into account when formulating the problem, it is also necessary to have a set of instances that can be used to validate said restrictions. However, there are few instances in the literature that deal with different restriction types, and those that do exist do not reflect their diversity. This is because most of these problem instances consider the orientation as the sole restriction.

Much of the research that studies a specific definition of the 3DPP does so considering its single-objective version, or by weighting the objectives. Hence, many of the instances proposed in the literature deal with the single-objective formulation of the problem. However, there is a real lack of compatible instances for the multi-objective formulation. To our knowledge, there are some sets of benchmark data for input minimisation problems when they are treated as single-objective instances of the 3DPP, and other sets for the output maximisation problems. For the input minimisation problems, we found the following data sets in the literature [222]:

- *Ivancic et al.* (1989) [128]: the objective is to minimise the number of containers used to pack all of the boxes. This benchmark has 47 instances of weakly heterogeneous items, denoted as *IMM*. These instances are used in the Single Stock-Size Cutting Stock Problem (SSSCSP).

- *Martello et al.* (2000) [157]: this set is for the Single Bin-Size Bin Packing Problem (SBSBPP), which packs strongly heterogeneous items. The data set includes 320 instances, which consider the orientation constraint.

- *Mack et al.* (2004) [153]: the data sets contains 100 instances to solve the Open Dimension Problem Weakly (ODP/W). These instances include the orientation of small items as a restriction.

- *Bortfeldt and Gehring* (1999) [37]: This set has 100 instances for a typical Open Dimension Problem Weakly (ODP/W) and Open Dimension Problem Strongly (ODP/S).

The following references are provided for the output maximisation problems:

- *Loh and Nee* (1992) [152]: a set of 15 problem instances, denoted as *LN1-15* instances, has been proposed for the Single Large Object Placement Problem (SLOPP). Each *LN* instance contains weakly heterogeneous boxes to be packed into a single container. The number of different types of small items varies between 6 and 10; while the total number of items varies between 100 and 250. Boxes may be left out of the packing solution if the single container is full.

- *Bischoff and Ratcliff* (1995) [29]: this set is for the Single Large Object Placement Problem (SLOPP), which packs weakly heterogeneous small items. Such a benchmark consists of a total of 7 classes in total, with each class including 100 instances, denoted as *BR1-7* instances.

- *Davies and Bischoff* (1999) [60]: this data set contains 8 classes, with each class including 100 problem instances, *BR8-15*. This set is merged with Bischoff and Ratcliff to yield the so-called *BR1-15* data set. This set is designed for the Single Knapsack Problem (SKP).

- *Bortfeldt and Mack* (2007) [42]: this set of problem instances is an extension of the *BR1-10* data sets, denoted as *BRXL1-10*. The difference with *BR1-10* is that the length of the container can be extended.

None of these instances consider the weight. The objective is focused only on maximising the container volume utilised. However, when the 3DPP is considered using a multi-objective formulation, like that given in Section 3.2, we need to know not only the volume of the container and the boxes, but their weights as well. But there are no instances in the literature that allow optimising more than one objective at a time. As a result, the need arises to develop an instance generator that can be used to create benchmarks that define, in addition to the volume of the container and the boxes, the weight associated with each box type and the container's weight limit. In addition, it would be interesting to know the optimal solutions in volume and weight, as that would help when evaluating the proposed heuristics.

For the multi-objective formulation, the only known instance that involves this problem is the one proposed by Dereli et al. [70]. The problem instance was collected from the company that distributed Procter & Gamble's products as well as their own products (paper, towels, toilet tissues, paper napkins, etc.). This instance included 12 different products. Each product has its quantities, dimensions ($length \times width \times height$) and weights of the boxes. The dimensions and maximum weight of the container are also known. However, we believe that a single instance is not enough for the development of a computational study. It is necessary to generate a large set of instances that allows comparing different approaches for the problem analysed herein.

4.3 Instance Generator

Doing an extensive study on the multi-objective 3D Packing Problem would require using a data set that would allow determining the efficiency of the methods presented in Section 3.4. As mentioned earlier, problem instances that deal with the 3DPP as a multi-objective formulation are non-existent. For this reason, we have developed software to generate multi-objective 3DPP instances, which we refer to as the Three-Dimensional Multi-Objective Generator (3D-MG). In our definition of the multi-objective 3DPP, we try to maximise the total volume and total weight at the same time. It is thus an essential requirement that the new instances provide not only the dimensions, allowed rotations, box types and the number of items of the different box types, but also the weight or profit associated with each. It is essential that the weight of the items generated not be necessarily proportional to their volume. In many real cases, maximising the total volume does not imply maximising the weight. That is, a box can have large dimensions and a lighter content than a smaller box. Thus, if we want to generate problem instances, we can take this feature into account in order to keep at least some degree of conflict between objectives.

4.3.1 Instance Generator Configuration

In order to define problem instances, the instance generator requires a set of input parameters. Considering the mathematical formulation give in Section 3.2, in the multi-objective 3DPP, we deal with a container with known dimensions ($L = length$, $W = width$, $H = height$), a maximum weight (P_{max}) and a set of N 3D rectangular boxes, where each piece belongs to one of the different box types $t_i \in \{t_1, \ldots, t_m\}$, with its own dimensions (l_i, w_i, h_i), profit p_i and rotation o_i. Considering this problem definition, the dimensions of the container (L, W, H) will form the first input parameters for the instance generator. The number of box types D will constitute another important input parameter for the generator. D specifies the total number of different box types that comprise each instance type. The instance generator

Input parameters	Definition
$L \times W \times H$	Dimensions of the container
D	Total number of piece types
D_s	Percent of piece types of D for Sol_1
D_l	Lowest dimensions for a box
D_h	Highest dimensions for a box
S_L	Stowage losses
W_G	Total weight gain of Sol_2
$W_p[]$	Percentage to calculate the weight of the box types of Sol_2

Table 4.1: Properties of the 3D-MG

allows box sets that can vary from small-sized to large-sized boxes. Thus, a set of boxes is identical or homogeneous when all the boxes are the same or very similar. If a set of boxes is weakly heterogeneous, then the problem instance has many boxes of each type but only a few types of different boxes. Finally, a set of boxes is strongly heterogeneous when there is only a few boxes of each type but many types of different boxes. To determine the dimensions of each box type, the generator requires two input parameters (D_l and D_h). These parameters determine the lowest D_l and highest D_h dimensions that a box type can have with respect to the dimensions of the container. It is thus possible to ensure that the dimensions of the box type do not exceed certain limits. Table 4.1 offers a summary of the input parameters that must be specified for the instance generator.

In order to obtain valuable instances for the multi-objective formulation of the problem, the problem generator designed is able to provide the optimal solution for each objective considered (volume and weight). Thus, the generator will give users the definition of a problem instance and two solutions for the problem instance: a solution that maximises volume (Sol_1) and one that maximises weight (Sol_2). These are needed to compare results and validate the quality of the methods proposed. Since a problem instance will have D different box types, a subset of M ($M < D$) types will be used to create Sol_1, and a subset of $D - M$ types will be used to create Sol_2. To determine the number of box types to be used in the solution Sol_1 we define the input parameter D_s. D_s is defined as the percent of box types out of the D that will be used to generate Sol_1. So $M = (D_s * D)/100$.

This way, two solutions are found (best in volume and best in weight) while the box types, and thus the problem instance, is being generated. To generate Sol_1, the generator completely fills the container, so the solution with maximum volume will have a total volume of $L \times W \times H$. The weight of Sol_1 will be equal to its volume, since the boxes used to create this solution will be assigned a weight proportional to the box volume.

To generate Sol_2, two additional input parameters must be specified in the generator, S_L and W_G (Table 4.1). The total volume of the container is slightly reduced due to stowage losses, S_L. The stowage loss is the difference between the freight volume and the container volume, which corresponds to another input parameter in the problem generator. This parameter can take on different values based on the container's capacity. S_L represents the length, width and height of the container that will remain unused for the creation of Sol_2. The input parameter W_G will represent the weight gain of Sol_2 with respect to the weight of Sol_1, i.e. W_G represents an increment to the container's overall weight of Sol_1. Since Sol_2 will have a lower volume than Sol_1 and a higher weight than Sol_1, the instances generated will maintain at least some degree of conflict between the objectives. The instance generator distributes the total weight gain W_G of Sol_2 plus a certain increment $W_p[]$ among the $D - M$ box types used for Sol_2. The weight of each box type is thus calculated by increasing the maximum weight P_{max} of the container by a certain percent, as determined by $W_p[]$. $W_p[]$ is a vector whose size depends on the total number of different pieces that must be created for Sol_2. Thus, for each box type $D-M$, we consider a different contribution to the maximum container weight P_{max}. The total contribution calculated for each piece type is finally divided by the total number of pieces available for each piece type $t_i \in D - M$, resulting in the exact weight of each piece type p_i.

4.3.2 Cutting Heuristics for Instance Generator

In order to decide on the particular dimensions and availability of the boxes used in Sol_1 and the boxes used in Sol_2, the instance generator applies two different cutting heuristics whose designs rely on the creation of levels. The levels create empty spaces which identify areas where different types of boxes can be placed. The levels are parallel guillotine cuts along the different axis $(x, y, and\ z)$.

With regards to the allocation of boxes within the container, the problem instances will be generated using the following assumptions:

- Each box is placed on the container floor or on top of another box.
- The loaded boxes can not overlap.
- The stability of the arrangement of the boxes is not considered.
- The levels are rigid, meaning it is not possible to generate levels between existing levels.

For the generation of Sol_1 and Sol_2, the cutting heuristics work in the same way. The number of box types that are created for each solution, Sol_1 and Sol_2, depends on D and D_s. For Sol_1, a maximum of $M = (D * D_s)/100$ box types can be created. Thus, M is the number of different box types available to build Sol_1, and its value depends on the value of the D and D_s parameters. The same applies to Sol_2, where $D - M$ box types are available for building Sol_2.

The cutting heuristics implemented work in three phases. In each phase, a different parallel guillotine cut is made along the x, y and z axes on the Cartesian system. The initial cuts will thus differ depending on the heuristic. So as to distinguish between the cutting heuristics, they will be referred to as Three-Dimensional Multi-Objective Generator in Length (3D-MGL) and Three-Dimensional Multi-Objective Generator in Depth (3D-MGD). If the 3D-MGL is applied, the cuts in the first phase are made along the y-axis, which corresponds to the width of the container, W. In the second and third phases, the cuts are made along the x-axis and z-axis, corresponding to the length L and height H of the container. In contrast, in the 3D-MGD heuristic, the first cut is made along the x-axis and the second along the y-axis, with the cuts being along the z-axis in the third phase.

To determine the number of cuts, c_i, generated in each phase and their dimensions, the heuristics keep track of a list of guillotine cuts for each of the phases. Each guillotine cut c_i generates a level with its associated dimensions (l_i, w_i, h_i) with $i \in [1, D]$. Each cut c_i is associated with a different piece type, meaning that each phase for solutions Sol_1 and Sol_2 can have a maximum of M and $D - M$ cuts respectively. When a cut is made, it is stored in the corresponding list, along with its dimensions and number of repetitions. The number of repetitions corresponds to how often the same type of cut is made along a given axis. When generating the different cuts, it is important to remember that two equal cuts belonging to different box types cannot exist, and that replicates of the same cuts do not necessarily have to be made consecutively. Once the heuristic finishes, the number of boxes available for each box type, and their respective dimensions and weights, can be determined.

4.3.2.1 Cutting Heuristic for Maximum Volume Solution

As noted earlier, the generator is designed to give one solution for each of the objectives studied in the definition of the 3DPP, volume and weight. In this section we explain the process of creating the different piece types for Sol_1, which build the solution with the maximum volume. Although there are two cutting heuristics, 3D-MGL and 3D-MGD, only the 3D-MGL will be explained in depth, since the other differs only in the order in which the cuts are made. With this in mind, the cutting heuristic in length (3D-MGL) generates Sol_1 as follows:

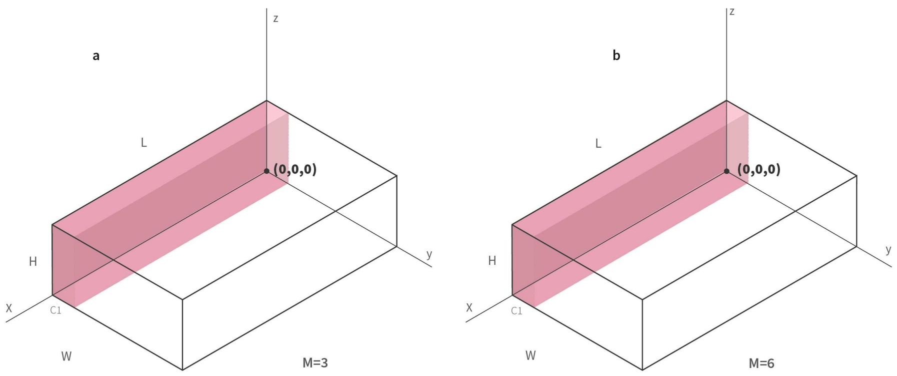


Figure 4.1: Creation first level in the width of the container

4.3.2.1.1 Phase 1

- The first cut c_1 is created by taking the back-left-bottom corner of the container as a reference, coordinates $(0, 0, 0)$ in the Cartesian system. The cut made along the x-axis - perpendicular to the y-axis - determines the width of the level (Figure 4.1). Since each cut is associated with a different piece type, this first cut corresponds to the width of the first piece type that is generated for Sol_1. The width of each cut is generated randomly based on the input parameters D_l and D_h. The first cut generated and its width are stored in $List\_w$, such that $List\_w = \{(c_1, w_1, n_1)\}$, where c_1 is the label of the first cut made along the y-axis, w_1 is its width and n_1 is the number of times the same cut is repeated across the y-axis (positive direction). Thus, $n_1 = 1$ when each cut is made for the first time. The cuts are numbered sequentially until the maximum number of box types corresponding to Sol_1, M, is reached.

- The next cut is made adjacent to the current one, c_{i+1}, and it is stored in $List\_w$ with its corresponding width, w_{i+1} (Figure 4.2). This process is repeated until one of the following exceptions occurs:

  – The maximum number of box types M is reached for Sol_1. By way of example, consider Figure 4.2-a, in which the value of $M = 3$. Thus, the maximum number of different cuts that can be made along the y-axis is 3. As shown in Figure 4.2-a there are three different types of cuts, whose widths are associated with the different piece types. The heuristic cannot continue to make more different cuts.

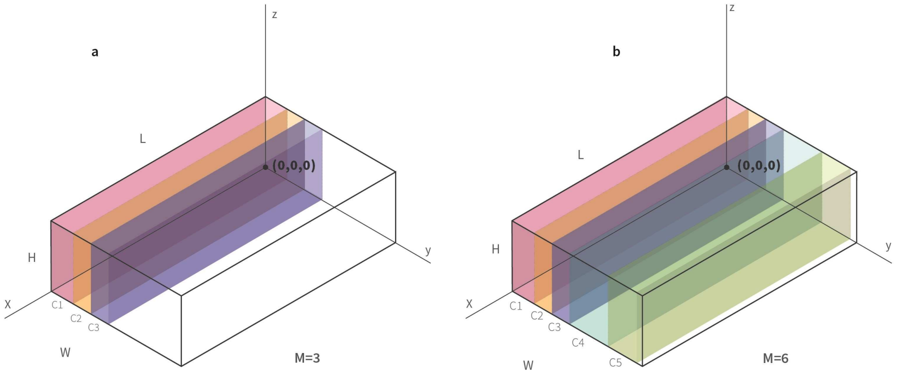


Figure 4.2: Creation new levels in the width of the container

- There is not enough space to continue creating new levels; that is, the unused space in the container is smaller than D_l, where D_l is the minimum size allowed for the dimensions of the different piece types. In Figure 4.2-b, the maximum number of cuts that can be made - for Sol_1 - is $M = 6$, but the width of the container (W) only allowed five different cuts to be made in this first phase, since the remaining space is not large enough ($< D_l$) to make another cut.

If the exception occurs because the maximum number of box types allowed for Sol_1 is reached, the algorithm must check if there is enough uncut space left to make more cuts across the y-axis (positive direction). At this point we should note that no new cuts are made; rather, cuts that replicate existing ones are made. This is because the number of different cuts c_i is determined by M in the case of Sol_1. If the remaining space is equal to or greater than D_l, the minimum size allowed to define the dimensions of the piece types, then the different cuts made will be replicated until the above stopping exception occurs. Every time a type of cut c_i is replicated, its n_i is increased. These cuts are made taking into account the order in which they were created, starting with the first cut made across the y-axis, c_1, and continuing with its adjacent cut (Figure 4.3). If a type of cut c_i cannot be replicated because its width w_i is longer than the space remaining in the y-axis, then new cuts of that width will not be attempted; instead, another cut is attempted until it is not possible to make any more cuts due to the lack of space across the y-axis ($< D_l$).

As Figure 4.3 shows, the cuts are replicated until the end of the y-axis is reached (W of the container) or until the space remaining is smaller than D_l. When it is not

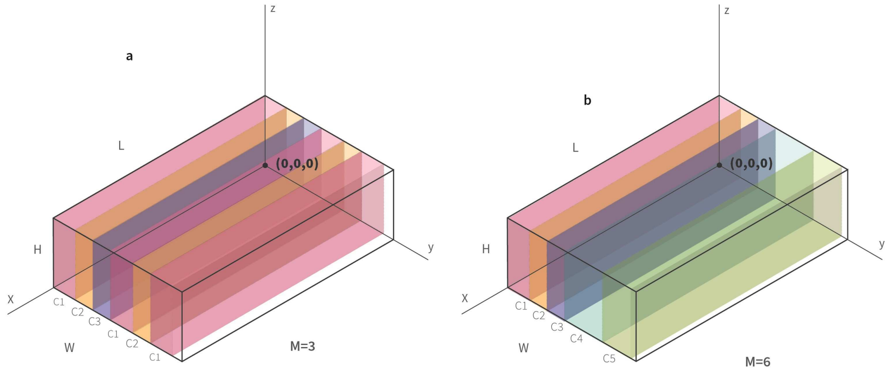


Figure 4.3: Filling the width of the container with replicated cuts

possible to make any more cuts, the algorithm must ensure that the width of the container (W) is fully occupied; that is, that the sum of all the widths of the cuts made across the y-axis (including replicas) is equal to W, such that $\sum_{i=1}^{p} w_i{*}n_i = W$, where p is the maximum number of cuts made in the first phase. Otherwise, if space remains in the y-axis (positive direction), then it must be completed until the upper limit of the axis is reached. We thus decided to allocate this space among all of the levels whose width matches those of c_1, which entails changing the value of the final width of the cuts associated with the first piece types, labelled as c_1 (Figure 4.4). Thus, for the cuts corresponding to c_1, their width w_1 is modified such that:

$$w_1 = w_1 + \frac{(W - \sum_{i=1}^{p} w_i * n_i)}{n_1} \tag{4.1}$$

Finally, the widths of the cuts in $List\_w$ are verified to be heterogeneous. As noted earlier, there cannot be two different piece types with the same dimensions. If this restriction is not satisfied, every cut c_i whose width w_i is equal or similar must be modified. This is done by locating all matching widths in $List\_w$ and modifying them. This process involves the following steps:

1. Look for all cuts (c_i, c_j) with equal width $w_i = w_j$. In the case of Figure 4.4-a, c_2 and c_3 have the same width ($w_2 = w_3$), meaning that the width of a certain cut is repeated twice.

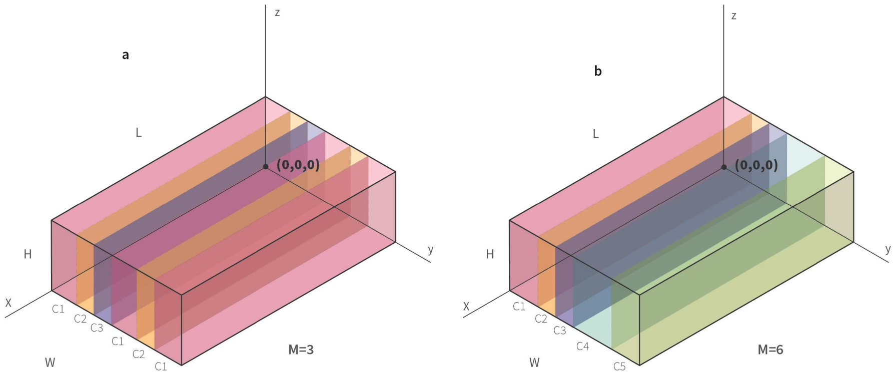


Figure 4.4: Completing the width of the container

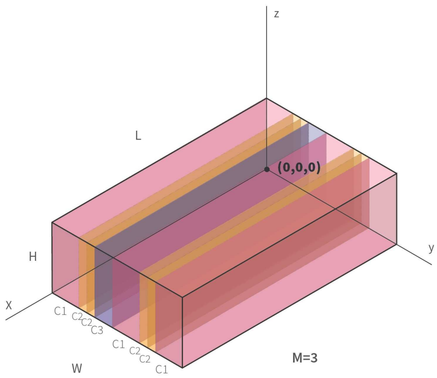


Figure 4.5: Final cuts for the width of the container

2. Starting with cut c_i where $i < j$ in $List\_w$, divide its width w_i as many times as w_i is present in $List\_w$. Thus, Figure 4.5 shows how the width of cut c_2 has been modified. In other words, the original cut has had its width modified by virtue of being subdivided once for each repetition of the cut that was present.

3. Once the width of a cut that did not satisfy the heterogeneity condition is modified, return to step 1 and look for cuts with the same width. Each repeated width is modified in each step of this process.

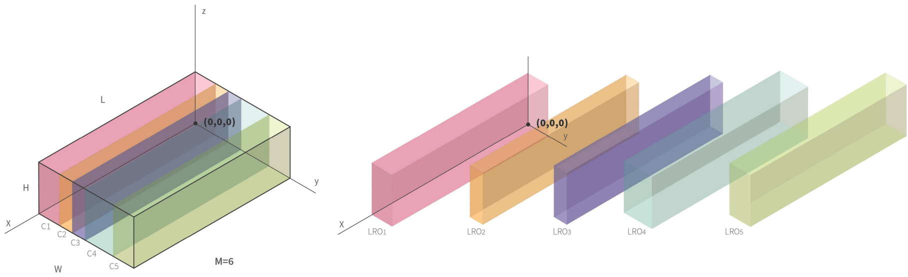


Figure 4.6: LROs from phase 1

4.3.2.1.2 Phase 2:

The different cuts made in phase 1 are regarded in this new phase as independent parallelepipeds to which new cuts can be applied. Each of these parallelepipeds, or Large Rectangular Objects (LROs), is associated with one piece type for Sol_1. Such is the case with Figure 4.6, which shows 5 large rectangular objects that correspond to the cuts made in the previous phase. Thus, the number of large rectangular objects corresponds to the size of $List\_w$. Every large rectangular object is regarded as a new space with its corresponding width calculated in phase 1, and its length and height corresponding to the dimensions of the container.

For this new phase, the process of creating the different piece types is initialised; that is to say, new cuts are made that will now determine the length of the various piece types, since their widths are now known. Moreover, the number of replicas that were generated for each large object during its creation phase must be considered in this new phase. Thus, a list ($List\_l$) will be maintained in phase 2 that will store the number of times that the same large rectangular object was replicated in phase 1 ($\{(c_i, w_i, n_i), \ldots, (c_p, w_p, n_p)\}$), its width, and the remaining data to be determined in the current phase. Phase 2 works as follows:

- The first cut is made in the large rectangular object whose back-left-bottom corner is at the Cartesian coordinates $(0, 0, 0)$ of the container. The cut that is made across the x-axis, parallel to the y-axis, determines the length of the level (Figure 4.7), and its dimension is determined based on input parameters D_l and D_h. The cut made across the x-axis is stored in $List\_l$ with its respective dimensions, w_i and l_i, now known. Thus, for the first cut c_1, $List\_l$ would be as follows: $List\_l = \{(c_1, l_1, w_1, n_{l_1}, n_{w_1})\}$, where c_1 corresponds to the first cut made in the existing large rectangular objects, l_1 to the length of the new cut and n_{l_1} to the number of its replicas along the x-axis. As noted earlier,

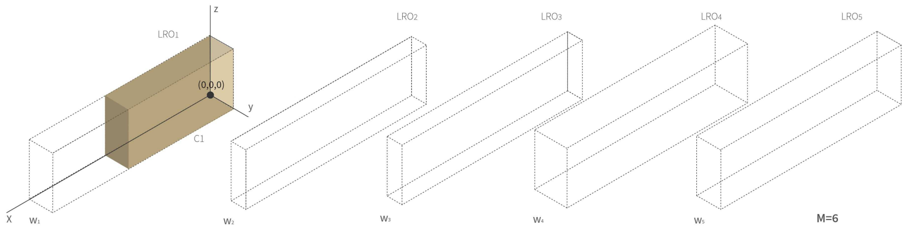


Figure 4.7: Creation of first level on the length of the first LRO

w_1 and n_{w_1} are values calculated in the previous phase related to a specific large rectangular object. When each cut is made for the first time, $n_{l1} = 1$. The cuts are numbered sequentially until the maximum number of box types in Sol_1 is reached, M.

- The next cut is made in the large rectangular object adjacent to the current one. If the last large object in $List\_l$ associated with the back-right-bottom corner is reached before the maximum number of piece types M for Sol_1, then new cuts are made, starting with the parallelepiped whose back-left-bottom corner coincides with that of the container. If new cuts cannot be made on a specific large rectangular object, then no more attempts are made at making cuts in said large object and the algorithm moves to the adjacent object. Every time a cut is attempted, it is stored in $List\_l$ with its respective dimensions and replicas, $List\_l = \{(c_1, l_1, w_1, n_{l_1}, n_{w_1}), \ldots, (c_q, l_q, w_q, n_{l_q}, n_{w_q})\}$, where q is the number of cuts made in the current phase. This process is repeated until one of the following exceptions occurs:

  – The maximum number of box types M is reached for Sol_1. By way of example, consider Figure 4.8, in which the number of piece types available for Sol_1 is $M = 6$, meaning only six different cut types can be made.

  – There is insufficient space to continue creating cuts in any of the different large objects; in other words, the unused space inside each large object is smaller than D_l .

If the exception occurs due to the maximum number of piece types M allowed for Sol_1 having been reached, the algorithm must then check if the entire length L of each rectangular object, which matches the length of the container, is being used. If it is not, and the leftover space in each large object is equal to or larger

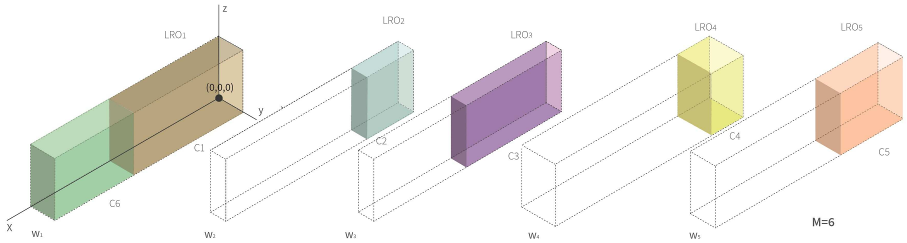


Figure 4.8: Creation new levels in the length of the LROs

than D_l, the minimum size allowed to define the dimensions of every piece type, them replicas of the existing cuts until the exception occurred must be made. We must realise that at this point, no new cuts will be made, but rather replicas of the existing cuts inside each large rectangular object. Only replicas of cuts previously made in each large object can be made inside them, as shown in Figure 4.9. A cut inside a large object that was created inside another large object cannot be made. Only cuts already contained beforehand in the specific large rectangular object can be created. Whenever a cut type c_i is replicated, its n_{l_i} is increased. These cuts are made considering the order in which they were created, starting with the first cut made along the x-axis of each large object and continuing with its adjacent object (Figure 4.9). As a result, the replicas must necessarily be generated consecutively. If any type of cut c_i cannot be replicated because the space remaining in the y-axis of a specific large rectangular object is smaller than its length l_i, then no more cuts of this type are attempted, and the next cut is tried. This same process is repeated in each of the large rectangular objects until no more cuts can be made due to a lack of necessary space in axis (remaining space $< D_l$).

When it is no longer possible to make more replicas of the various cuts c_i in each of the large objects, the process continues as in the previous phase. The heuristic ensures that for each large object, its length L is completely filled. In other words, the sum of all lengths of the cuts made across the x-axis (including replicas) is equal to L for each rectangular large object, $\sum_{i=1}^{q} l_i * n_{l_i} = L$, where q is the maximum number of different cuts made in total in the rectangular large objects in this phase. If there is space remaining in the x-axis (positive direction) for some of the rectangular large objects, it must be completed until the upper limit of the length (L) of the container is reached. This case is shown in the first and third rectangular large objects in Figure 4.9. This implies modifying the value of the final length of each of the associated cuts (Figure 4.10). Phase 2 ends with a check of

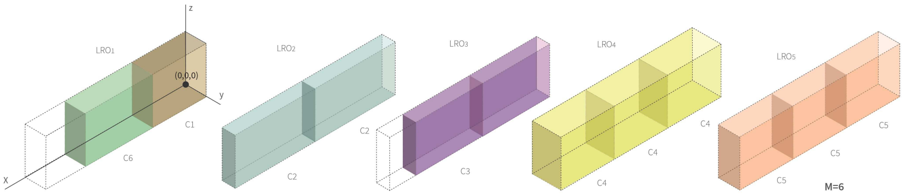


Figure 4.9: Filling the length of the LROs with repeated cuts

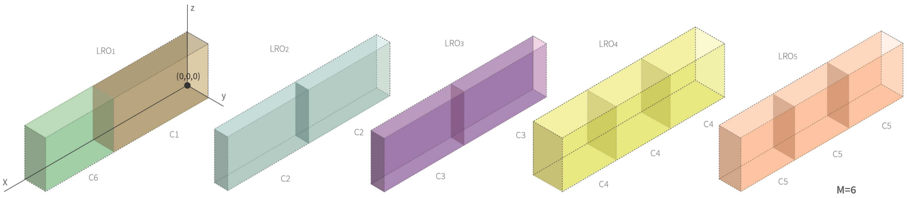


Figure 4.10: Completing the length of the LROs

the heterogeneity of the lengths created for the different levels. Figure 4.11 shows the case in which cuts c_2 and c_3 are the same, as are c_4 and c_5. This process entails carrying out the following steps:

1. Look for all cuts (c_i, c_j) with equal length $l_i = l_j$. In the case of Figure 4.10, c_2 and c_3 have the same length ($l_2 = l_3$), which implies that a certain l_i is repeated twice. The same thing happens with c_4 and c_5, which have the same length ($l_4 = l_5$).

2. Starting with the first cut c_i parallel to the y-axis in $List\_l$, where $i < j$, divide its length l_i as often as there are repetitions of this same length l_i in $List\_l$. Thus, in Figure 4.11 we see how the length of cut c_2 has been modified. That is, the length of the original cut has been modified by being subdivided as often as it was repeated in the list. The same thing happens with cut c_4.

3. Once the length of one of the cuts that did not satisfy the heterogeneity restriction is modified, the heuristic returns to step 1 and looks for cuts with the same length. Each repeated length is modified in each step of this process. A single side is modified in each iteration.

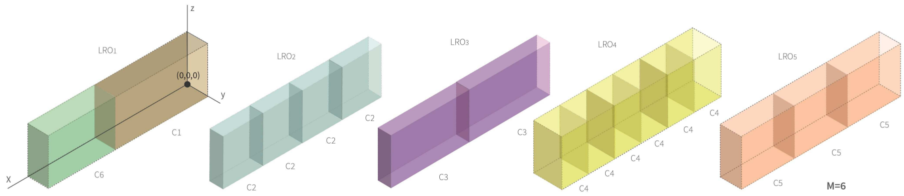


Figure 4.11: Final cuts for the length of each LRO

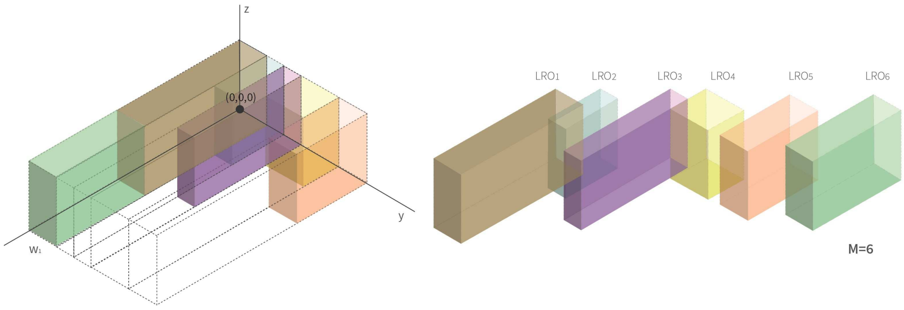


Figure 4.12: LROs from phase 2

4.3.2.1.3 Phase 3:

Phase 3 involves the large rectangular objects created as a consequence of the cuts made in the previous phases. Each large object corresponds to an empty space where new cuts can be made. These cuts will then be associated with the different piece types to be created for Sol_1. Figure 4.12 shows the set of large objects resulting from the cuts made in phases 1 and 2, where $M = 6$. As we can see in this figure, there is a total of 6 different large objects, each with a length l_i and a width w_i, both known, and a height h_i, which corresponds to the height of the container H. Thus, in this new phase the number of LROs to be considered is equal to the number of different cuts, q, that were able to be made in phase 2 (size($List\_l$)).

In this new phase, as in the previous one, new cuts are made that will define the final dimensions of the various piece types. In this phase, the still unknown dimension will be determined, namely the height of the various piece types, since their length and width are already known. For this phase, a list will be maintained, $List\_h$, where the data collected since phase 2 ($\{(c_i, l_i, w_i, n_{l_i}, n_{w_i}), \ldots, (c_q, l_q, w_q, n_{l_q}, n_{w_q})\}$), will be stored, along with those that will be defined in the current phase. Phase 3

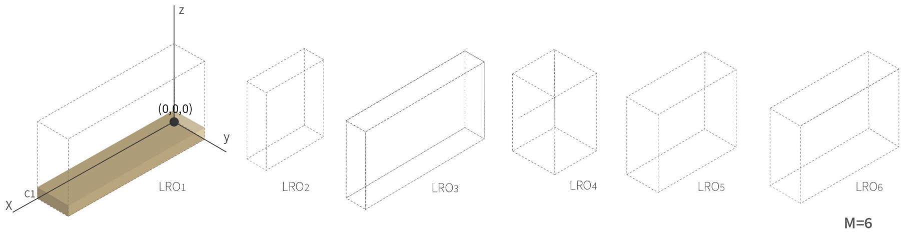


Figure 4.13: Creation of first level on the height of the first LRO

works as follows:

- The first level involves the large rectangular object that is at the $(0,0,0)$ point of the container. The cut made across the z-axis at each of the large objects determines the height h_i of each piece type for Sol_1. This initial cut is stored in $List\_h$, such that $List\_h = \{(c_1, l_1, w_1, h_1, n_{l_1}, n_{w_1}, n_{h_1})\}$, where c_1 is the new cut generated along the z-axis, (l_1, w_1, h_1) correspond to the dimensions of the first piece type, and $(n_{l_1}, n_{w_1}, n_{h_1})$ define the number of times that the level has been repeated in its associated axis, where n_{h_1} is the number of times that the new cut is repeated during the current phase. The cuts are numbered sequentially until M is reached.

- The remaining cuts are made in the large rectangular objects adjacent to the current one, taking into account the order in which they were created in the previous phase. The initial cuts in each of the large objects are made taking the floor of the container as a reference (Figure 4.13). As in phase 2, when the last large rectangular object is evaluated, the process starts at the large object whose corner is at point $(0,0,0)$ in the container. Those large objects in which new cuts cannot be made are ruled out. Every time a cut is made, it is stored in $List\_h$ with its respective dimensions and replicas, $List\_h = \{(c_1, l_1, w_1, h_1, n_{l_1}, n_{w_1}, n_{h_1}), \ldots, (c_z, l_z, w_z, h_z, n_{l_z}, n_{w_z}, n_{h_z})\}$, where z is the number of cuts made in the current phase. This same process is repeated until one of these exceptions occurs:

  - The maximum number of box types M for Sol_1 is reached. By way of example, consider Figure 4.13, in which the number of piece types available for Sol_1 is $M = 6$, meaning only six different types of cuts can be made.

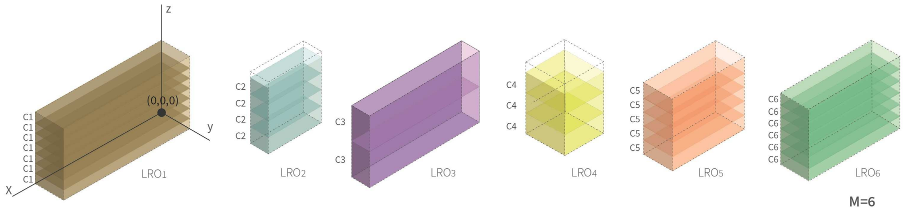


Figure 4.14: Filling the height of each LRO with repeated cuts

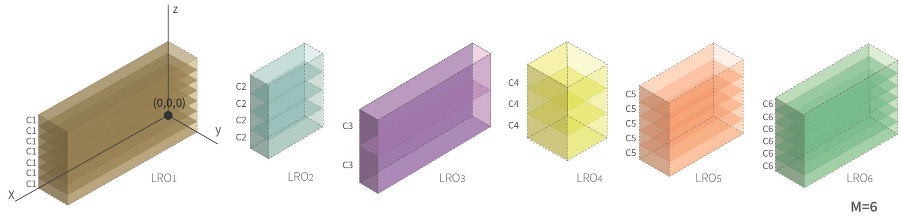


Figure 4.15: Completing and final cuts for the heigth of each LRO

– There is not enough space to continue making new cuts in any of the various large objects, meaning the unused space inside each large object is smaller than D_l ($< D_l$), where D_l is the smallest size allowed for the dimensions of the various piece types.

The rest of the phase follows the same procedure as the previous ones, this time considering the z-axis and the height H of the container. The first check made is to verify that the height H of each large rectangular object is completely filled. Figure 4.14 shows how the second and fourth large rectangular objects do not satisfy this condition. To fill in the remaining space in these large rectangular objects, the same procedure explained for the previous phases is used. Finally, Figure 4.15 shows the final condition after filling in the remaining space in the z-axis. Note that the heights of the cuts made are all different, thus complying with the heterogeneity condition. Thus it is not necessary to modify the height of the cuts in order to make them heterogeneous.

The number of piece types generated for Sol_1 is equal to M. After the different cutting heuristics (3D-MGL and 3D-MGD) are applied, though, it may not be possible to generate M different types of pieces. As a result, the remaining piece types that could not be created using the heuristics will be generated randomly.

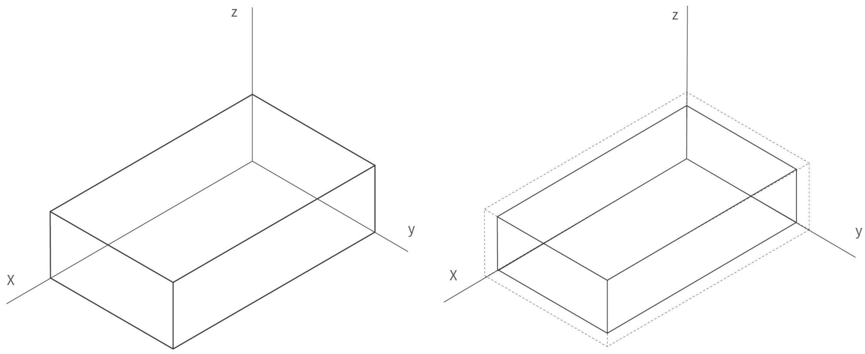


Figure 4.16: Applying S_L to Sol_2

When the phase 3 of the heuristic is completed, the information needed to determine the dimensions, availability and weight of each box type will be available. The number of box types $t_i \in \{t_1, \ldots, t_m\}$ and their dimensions are associated with each of the large rectangular objects obtained at the end of phase 3 of the cutting heuristic. Thus the number of pieces available for each piece type will be determined by the number of replicas stored in $List\_h$ for each type of cut c_i in the last phase of the cutting heuristic. The number of boxes for a given piece type t_i associated with a cut c_i is therefore $b_i = n_{l_i} * n_{w_i} * n_{h_i}$. To generate Sol_1, the weight associated with each piece type is proportional to its volume, $p_i = l_i * w_i * h_i$.

4.3.2.2 Cutting Heuristic for Maximum Weight Solution

To generate Sol_2, the cutting heuristic uses the same procedure but considering $D - M$ pieces available instead of D. The other difference for Sol_2 is that the cutting process does not start from the whole container. It begins the partition process from a smaller container $((L - S_L) \times (W - S_L) \times (H - S_L))$ in order to yield a solution with a lower volume than Sol_1, but a higher weight (Figure 4.16). The weights of the box types associated with Sol_2 are not proportional to their volume, as was the case for Sol_1. The weights of the different piece types t_i that make up Sol_2 are calculated by considering the weight of the container obtained in Sol_1 and increasing it by a percentage $W_G(\%)$. Based on this, the weights of the different piece types t_i in solution Sol_2 are calculated. Each piece type t_i has associated with it a given percentage that determines a certain increase over the weight of the Sol_1.

$L \;\; W \;\; H$	Dimensions of the container
P_{max}	Maximum weight of the container
D	Number of different box types
$l_0 \;\; or_1 \;\; w_0 \;\; or_2 \;\; h_0 \;\; or_3 \;\; p_0 \;\; b_0$	length orientation1 width orientation2
.	height orientation3 weight and number
.	of boxes of type t_i
.	
$l_i \;\; or_1 \;\; w_i \;\; or_2 \;\; h_i \;\; or_3 \;\; p_i \;\; b_i$	

Table 4.2: Output file format

These percentages are passed as an input parameter $W_p[]$ to the instance generator. W_p is a vector of percentages whose size is equal to $D - M$. When phase 3 of the cutting heuristic to built the maximum weight solution ends, the number of pieces available for each piece type is known. The weight of each piece type is thus calculated according to the following steps:

1. The total weight calculated for Sol_1 is increased by W_G.

2. For each piece type in Sol_2, its total weight is calculated using $W_p[](\%)$.

3. The total value of the weight calculated in the previous step is divided by the number of pieces b_i available for each piece type t_i, yielding the value of the weight p_i for each piece type.

4.3.3 Output File from Instance Generator

As a result of executing the instance generator, an output file is obtained. Table 4.2 shows the content and format of the output file. Each output file specifies the dimensions of the container (L, W, H). A maximum weight P_{max} (specified by the total weight of Sol_1 increased by W_G), and a set of D different piece types, each one formed by b_i rectangular boxes. Each type of box is characterised by its dimensions (l_i, w_i, h_i), profit or weight p_i, a demand b_i and a number of orientations allowed $o_i \, \epsilon \, [0, 5]$. Thus, o_i is the value that results from applying $2 * or_1 + 2 * or_2 + 2 * or_3$. According to the values taken in $or \in [0, 1]$, the boxes may rotate on any or all sides.

ISO designation	Common Name	Internal Dimensions		
		Length (L)	Height (H)	Width (W)
1EEE	45 foot high cube	13.542 m	2.655 m	2.330 m
1EE	45 foot high	13.542 m	2.350 m	2.330 m
1AAA	40 foot high cube	11.998 m	2.655 m	2.330 m
1AA	40 foot standard	11.998 m	2.350 m	2.330 m
1A	40 foot	11.998 m	2.197 m	2.330 m
1BBB	30 foot high cube	8.931 m	2.655 m	2.330 m
1BB	30 foot standard	8.931 m	2.350 m	2.330 m
1B	30 foot	8.931 m	2.197 m	2.330 m
1CC	20 foot standard	5.867 m	2.350 m	2.330 m
1C	20 foot	5.867 m	2.197 m	2.330 m
1D	10 foot	2.802 m	2.197 m	2.330 m

Table 4.3: Operational values for containers

4.4 Problem Instances

In order to generate a set of problem instances for the multi-objective 3DPP, we have selected possible values for the input parameters. Determining possible values for the input parameters requires knowing more about the containers used in the industry. Industrial containers are made according to the ISO (International Organization for Standardization) standard, specifically, ISO-688, which contains the main characteristics of industrial containers, including the dimensions, recommended volume and standard gross weight of series 1, or freight, containers [183]. Containers are generally made of steel, but there are some that have been made using materials such as aluminium.

One of the characteristics employed when selecting containers is their dimension. When dealing with freight containers, we must differentiate between their internal and external dimensions. The external dimensions are needed to determine the mode of transport used to move the container from one place to another. The internal dimension is the one that really matters when dealing with loading problems, and it relates to the dimensions of the largest free rectangular parallelepiped that can be inscribed inside the containerr [12]. Table 4.3 shows the operational values for the containers specified in the *ISO UNE 49750 Standard*.

Other important concepts when dealing with the problem of loading containers include the tare, gross weight, recommended use volume, net weight or maximum recommended loading weight and loss factor. The gross weight is the total weight

Stowage losses by container type	
Container type	Stowage losses
10-foot ISO (3m)	17%
20-foot ISO (6m)	12%
30-foot ISO (9m)	10%
40-foot ISO (12m)	8.9%

Table 4.4: Stowage losses by container type

of the empty container (tare) plus the freight. A container's tare depends on the construction material. Another important aspect is the effective volume available inside the container. We thus differentiate between the useful volume and the effective volume. The useful volume is the general capacity that the container can store, whereas the effective volume is the volume that can be taken up by the freight inside the container. The difference is called the recommended volume, the maximum calculated volume for a fully laden container. The net weight or maximum load is the recommended weight that can be loaded inside each container. There is also a stowage loss factor, which accounts for the difference between the freight volume and the container volume. Table 4.4 shows the different recommended values that this parameter can take depending on the container type [183].

Next we describe the benchmarks generated to study the 3DPP as a multi-objective optimisation problem, the goal being to maximise the freight volume and weight simultaneously in a single container. The instances were generated by the 3D-MGL filling heuristic, using different values for the input parameters in an effort to study how these parameters influence the behaviour of the filling heuristics explained in Section 3.4.3.

So as to study instances that more closely adapt to reality, we began the study by generating instances in which the container's internal dimensions correspond to two of the most widely used container types, both described in the *ISO UNE 49750 Standard*. The same does not apply to the size of the boxes, which are generated completely at random by the 3D-MGL heuristic utilised by the instance generator. The containers defined in Table 4.3 have been used. Every container shown in this table is defined in the industry as a series 1 freight container, or large container [183]. Series 1 containers include: flat rack, open top, open side, dry van, high cube, reefer, tanker, and flexi-tank. Series 1 containers are classified as type A, B, C, D, AA, BB and CC [183]. The type *1C* (6m) and *1A* (12m) are some of the most widely used in trade worldwide and are used to transport bulky, heavy cargo. This is why both container types were selected for the generation of instances. We have generated 36 different problem instances for each container type. The dimensions of the containers are determined by the ISO. We thus have a type *1C* container whose measurements are $587cm \times 233cm \times 220cm$ for length (L), width (W) and height (H) respectively, and a type *1A* container with a loading capacity of $1200cm \times 233cm \times 220cm$. The volume of the containers is $L \times H \times W$. The box set of the different instances vary from identical or homogeneous to large-sized boxes. To determine the box sizes, the generator requires two input parameters (D_l and D_h). These parameters determine the smallest and largest dimensions that the box can have with respect to the dimensions of the container. The set of smallest and largest dimensions used was the following: $[D_l - D_h] = [5 - 10]\%, [15 - 20]\%, [25 - 30]\%$. A box is considered to be small-sized when its dimensions are between $[5-10]\%$ with respect to the length, height and width of the container. It is medium-sized when $[15 - 20]\%$ values are used, and large-sized when its size ranges between $[25 - 30]\%$.

So as to study the behaviour of the heuristics when evaluating instances with identical, weakly heterogeneous and strongly heterogeneous boxes, we generated problem instances with 2, 5, 10 and 15 different types of boxes. To generate instances, we need two more parameters: the stowage losses (S_L) and weight gain factor (W_G). The S_L (in %) is defined as the difference between freight volume and container volume. This value takes a percentage depending on the capacity of the container. For a type *1C* container, the stowage loss is 12%, and for a type *1A* container, it is equal to 8.9%, as indicated in Table 4.4 [183]. The parameter W_G (in %) determines the maximum weight increase of the container. This increase is expressed as a percentage. Three different values were used for the weight gain factor ($W_G = 10\%, 20\%$ and 30%), so three different set of instances were obtained based on the WG. This parameter enhances the multi-objective approach of the problem. The dimensions and weights of the boxes are randomly generated.

Several percentages were used to calculate the weights $W_p[]$ of the piece types corresponding to Sol_2: $\{15.0\%, 25.0\%, 45.0\%, 55.0\%\}$. Note that not all of the percentages are used for every instance, since they depend on $D - M$, with $M = (D * D_s)/100$.

Appendix C shows the set of variable values given to the instance generator as input parameters. In both tables we can see te maximum value of the two objectives, volume and weight. Based on these values, it is simple to determine how efficient the heuristics developed to decode the individuals. This new data set is called *1C* and *1A*, in reference to the container type as per the ISO standard. Thus, for container type *1C*, the notation $1C\_01$ to $1C\_36$ would be used, while for container type *1A*, it would be $1A\_01$ to $1A\_36$. Figure 4.17 shows the output files for some generated problem instances.

```
587 233 220
3.91165e+07
2
117.4 0 46.6 0 44 1 240717 125
101.191 0 44.736 0 42.24 1 78233 125
```
(a) 1C-06

```
1200 233 220
7.99656e+07
2
240 0 46.6 0 44 1 492096 125
197.564 0 44.736 0 42.24 1 159931 125
```
(b) 1A-06

```
587 233 220
3.91165e+07
5
97.8333 0 40.2843 0 36.6667 1 144509 72
117.4 0 39.846 0 44 1 205828 50
48.9167 0 36.3697 0 18.3333 1 32616.6 288
93.92 0 36.3079 0 35.2 1 162985 108
112.704 0 38.2521 0 42.24 1 286854 75
```
(c) 1C-15

```
1200 233 220
7.99656e+07
5
240 0 52.8447 0 44 1 558040 50
120 0 42.0816 0 22 1 111095 200
80 0 43.1476 0 14.6667 1 50626.5 225
230.4 0 47.6278 0 42.24 1 479794 75
115.2 0 40.3983 0 21.12 1 219905 200
```
(d) 1A-15

```
587 233 220
3.91165e+07
10
96.8 0 37.7053 0 36.6667 1 135257 36
97.8333 0 38.3691 0 18.3333 1 68819.2 72
117.4 0 39.7212 0 44 1 205184 25
48.9167 0 35.8679 0 12.2222 1 21444.3 216
110.6 0 42.8256 0 22 1 110610 50
32.6111 0 38.5111 0 9.16667 1 11512.3 432
93.92 0 35.3833 0 35.2 1 135821 72
46.96 0 36.8343 0 17.6 1 20373.2 288
112.704 0 38.1323 0 42.24 1 704097 25
56.352 0 41.1125 0 21.12 1 58674.8 100
```
(e) 1C-24

```
1200 233 220
7.99656e+07
10
251.717 0 71.1955 0 44 1 788529 15
200 0 35.693 0 36.6667 1 261749 36
240 0 40.1749 0 22 1 212124 50
120 0 42.2417 0 44 1 223036 50
80 0 43.6948 0 14.6667 1 51268.5 225
222.425 0 35.5978 0 11 1 87096.2 80
230.4 0 55.1474 0 42.24 1 399828 50
192 0 34.2653 0 35.2 1 333190 36
115.2 0 38.5679 0 21.12 1 359845 100
230.4 0 40.552 0 14.08 1 159931 75
```
(f) 1A-24

Figure 4.17: Output file examples

Chapter 5

Graphical User Interface

This chapter presents a Graphical User Interface (GUI) for a service used to visualise the 3DPP as applied to the logistics industry. The goal is to develop an advanced GUI that can be used to display the packing process within a large object. In this context, we present a study on the state of the art in the field of three-dimensional graphics, in particular for use on websites. We then show a comparison of those libraries that might be regarded as interesting 3DPP problem. Then, in an effort to ascertain the most-used features, we evaluate some interfaces available online for the problem at hand. Finally, we present the Three-Dimensional Graphical User Interface (3DGUI) developed to display both the starting instances and the results obtained when the different filling heuristics are applied to solve the multi-objetive 3DPP.

5.1 Introduction

The Graphical User Interface (GUI) has been an important component in the evolution of technology on human-computer interfaces [162]. The GUI is one of the main factors in the success of many computer tools and applications, since it allows users to forego complex tasks, such as issuing commands via text. It should thus come as no surprise that the GUI has dominated Human-Computer Interaction (HCI) for over two decades. This interaction, which allows users to carry out complex tasks, is based on visual relationships represented by windows, icons, menus and pointers. This is why GUIs are also known as WIMP: Windows, Icons, Menu and Pointers. The comfort and convenience provided by these interfaces has resulted in their widespread use. However, the increased power of many graphics systems and processing capacities has caused interface technology to advance toward a new generation of GUI. Dedicated and specialised mathematics processors 3D have resulted in more natural HCIs. This new generation is intended to use the resources and capabilities of current computer systems more efficiently. Thus, the Three-Dimensional Graphical User Interface (3DGUI) was created, which in many settings has replaced the old Two-Dimensional Graphical User Interface (2DGUI). In the literature, the three-dimensional interface is used to describe a wide variety of interfaces for displaying or interacting with three-dimensional objects [167]. However, many current user 3DGUI are hybrids between 2D and 3D interfaces [168]. The visual elements employed in a 3DGUI are based on the Euclidean plane $\Re^3$, meaning they are three-dimensional objects located in (x, y, z) space.

The term 3D graphics refers to any element that can be displayed on the Euclidean plane $\Re^3$. If extrapolated to the computing industry, 3D graphics are those that with help from specialised software that, through mathematical calculations applied to three-dimensional geometric objects, yields a projection that appears to be 3D by making use of a visual projection in 2D. The process of creating computer-generated 3D graphics has three phases [21]:

- Modelling: process of developing a mathematical representation of any surface or object using a set of points in a three-dimensional space connected with geometric shapes like triangles. The objects are shaped during the modelling stage for subsequent use in the scenery. This process can include a description of the object's surface, such as with textures, and the properties of the material and other characteristics [189].
- Arrangement of the scenery: in this stage, the objects, lights, cameras and other elements that will be used to produce the image or animation are arranged. Lighting is a key aspect when composing a scene, because it contributes to the aesthetic result and requires a physical understanding of the light in order to recreate it accurately. In this area, we can differentiate among four basic types of lights [104]: point light, infinite light, spot light and image-based light. The point light is also known as an omni light. This type of light emits with the same intensity in every direction. These types of lights have a position, but no direction. The infinite light is also known as a directional light. This type of light has a direction, but no defined position. It thus emits light parallel to a certain direction. Spot lights simulate the light given off by a desk lamp. They have defined positions, directions and a hotspot angle. Finally, an image-based light provides an environmental light source, where the light does not have a point, but it does have a texture map, spherical panorama.
- Rendering: is the final process that generates the image or animation from the scenery created. The term render is a calculation process that entails complex computer operations intended to generate an image 2D based on scene 3D.

This results in an interpretation from scene 3D to its projection as a two-dimensional image 2D.

In the computer industry, this procedure is used for very different purposes, from creating videogames to displaying data of various types. The technologies that can be used and the contexts where interfaces that rely on three-dimensional graphics can be displayed are also very different. There are numerous examples of 3D design software, such as Blender [97], which is open source, or the Unity engine [11]. The models generated with them can even be exported and used on web platforms, which gives programmers many possibilities and enhances the interactivity of the contents.

5.2 3D Graphics in the Web

Web technology has evolved to meed user demand for better performance and features. At first, web technology was developed using static programming languages, which then evolved into the programming languages used today to develop the dynamic web. One of the current pillars of web programming, JavaScript, was developed by programmer Brendan Eich [176]. This language was born of the need to create websites where a human-machine interaction could be generated. As a result, JavaScript is used primarily to generate dynamic web pages.

Initially, HyperText Markup Language (HTML) provided the ability to control the look of websites. However, the growing differences among its implementation in various browsers made it difficult for the same web interface to have the same appearance in every browser, which resulted in Cascading Style Sheets (CSS). Thus, while HTML allows structuring the content of the site, CSS allow separating the structure from the presentation. In other words, CSS define the appearance of every element that comprises the web interface, treating them separately from how the website's structure is defined. Initially, these were some of the tools that were used to play animations and graphics natively on the web. Other technologies have since appeared that can be used to display 3D graphics on web interfaces, such as vector graphics, the images of which consist of vectors joined together. It soon became obvious, however, that this system was not sufficiently practical, so work began on more refined techniques, like the canvas element in HTML5 to generate dynamic images, and the Web Graphics Library (WEBGL) specification. WEBGL is an Application Programming Interface (API) for rendering 3D interactive graphics from 2D on any compatible website without the need for plug-ins. The introduction of WEBGL has expanded the development of interactive 3D content through the use of web technology. Since WEBGL is a low-level specification that works directly with the Graphics Processing Unit (GPU), its coding is more complex in comparison to other web standards. As a result, numerous libraries have emerged that allow exploiting the features of WEBGL technology in a simpler fashion. The most important are [14]: *CDL*, *CURVED*, *CubicVR*, *CopperLicht*, *GLGE*, *OD*, *OSG*, *PhiloGL*, *SceneJS*, *SpiderGL*, *TDL*, *Three.js*, *XDOM*, *EnergizeGL*, *GammaJS*, *GTW*, *JSD*, *Kuda*, *BabylonJS* and *Pred*. Included among these libraries is *Three.js*, which has sparked the most interest among web developer.

5.3 3D Graphics Engines

In an effort to implement a 3D graphical user interface using web technology that allows displaying the results output by the different heuristics displayed in Section 3.4, we considered some of the most versatile and powerful libraries that enable the human-machine interaction. Our study of the main libraries was carried out with a fixed goal, that of being able to display in a simple manner the process by which a small rectangular item is placed within a large rectangular object (container). To this end, we must realise that the libraries give the option of working with transparencies or grids that make it possible to visualise the process. We also wish to be able to interact with the web application by using a mouse so as to see the container and its contents from different angles. We would also like to change the location of the small rectangular items contained inside the large object so that the user can reposition them at will. The following libraries were evaluated:

- *CubicVR 3D*: this library allows easily adding mouse controls, resulting in fairly simple code. However, even though it offers transparencies and grids, the former do not allow regulating the number of divisions for the large object's various dimensions. In the case at hand, these would be the length, height and width of the container. Only the number of divisions common to every dimension can be specified.
- *GLGE*: the scene is coded in an eXtensible Markup Language (XML) file. The JavaScript code is loaded, the relevant variables for the camera, scene and so on are assigned and the scene is rendered. However, it has the drawback that it is difficult to increase the presence of small items, not only because it is hard to add them, but because *GLGE* does not help the user create them. This means that all of the points have to be defined and the pertinent calculations done to place them in the proper location.
- *SceneJS*: allows for the use of transparency. When grid material is used, however, its divisions cannot be adjusted.

- *PhiloGL*: presents more drawbacks than advantages for the problem at hand, since there is no grid and the transparencies do not work altogether well, since we were not able to apply them to individual elements.
- *Three.js*: this framework easily allows mouse controls and placing new items in the scene. On top of this, transparencies y grids can be used, which provides for independent control of the divisions of the latter.

Based on the above, we determined that the *Three.js* library was the best option for displaying the results involved in this work. It can be used to create 3D graphics and animations in web platforms in real time with sufficient quality. *Three.js* is a fairly powerful and efficient JavaScript library that uses the WEBGL specifications included in HTML5. One of the features that has made *Three.js* so popular for web development is that it lets the developer ignore many of the properties of WEBGL by hiding the complexities of programming in a low-level language. This simplifies development greatly and makes it easier to manipulate its elements. *Three.js* offers several easily manipulated elements. The ones that are most relevant and useful for our purposes are:

- Scenes: allows adding and deleting objects in execution time. They contain elements as essential as the camera, the mesh and the light. The camera defines the user's point of view and can be used to control actions like zooming, the position and more. *Three.js* offers two types of projections:
  - Parallel projection, which maintains the size of the objects, independently of their distance from the camera.
  - Conic projection, which distorts objects based on their distance to and position from the camera, as in the real world.
- Renderers: Canvas, SVG and WEBGL.
- Geometry: list of vertices and faces that comprise an image. *Three.js* offers a library of predefined polyhedrons, such as planes, cubes, 3D text, etc.
- Objects: meshes, sprites, bones, etc.
- Materials: can be used to describe the appearance of objects, textures, shadows and colours, among others.

It is thanks to these characteristics, along with its ease of use, that we opted to use it as the API for developing the software that will allow us to display the solutions to the 3DPP. *Three.js* is also very well documented. In our implementation of the GUI, we settled on JavaScript along with *Three.js*.

5.4 Software to Display the 3DPP

Due to the prevalence of the problem in question, 3DPP, in many logistics process in industry, a wide variety of software has been developed to help visualise loading processes considering some of the restrictions addressed in Section 2.4.1. Most of these applications are developed and distributed by software companies, normally under proprietary licences. There are usually trial versions of these programs available, but their features are constrained. The number of non-commercial tools is very limited and their feature set much more restricted than those offered in commercial products.

The software developed for loading and packing problems provides users with alternative methods for entering the data sets that define the problem's properties (dimensionality, item shape, type of cut, etc.), and even for displaying the solution arrangements. In an effort to develop a web interface for the problem considered in Section 3.2, we studied each of the features offered by applications already in existence to determine all of the features that could be relevant and applicable to the development of a specific GUI for the multi-objective 3DPP. Some of these software packages are described below:

- CargoWiz [4]: is specialised software for loading trucks and containers that calculates the cargo, weight and space needed for a shipment. The loading process takes into account a set of priorities, like sequential deliveries or grouping of items. It also allows a set of orientations for each item type, it knows not to place heavy boxes atop lighter ones, and it considers stacking limits for cargo items.
- CargoManager [3]: this program can be used to plan and see the cargo for thousands of containers. It helps maximise space use in containers, pallets or any other type of rectangular space. It takes into account several practical loading constraints, like the cargo's weight and fragility.
- MaxLoad Pro [9]: is optimisation software that plans the loading of trucks and containers. It determines how to load various types of products of different sizes and on different scales. It starts with the boxed product, it then moves the boxes on pallets and then the pallers are loaded into trucks. When loading items, it considers the stacking requirements, loading orientation and placement, as well as palletization preference.
- AutoLoad Pro [2]: automatic optimisation software for loading cargo into trucks, containers, cartons and pallets. It uses a two-state optimisation technique. The loading and display considers restrictions such as the orientation, stacking strength and safety, loading sequence and item grouping, and it optimises the weight and volume use inside the container.

- PacKVol [10]: this cargo planning software is designed to make the best use of space in containers and trucks, helping to reduce transport costs. It relies on an intuitive graphical user interface an it features manual and automatic modes. The automatic mode offers fast solutions to problems of load plan optimisation, while manual mode allows modifying or creating the entire load plan.
- CubeMaster [6]: load planning, optimisation, diagramming and distribution software for containers, trucks, pallets, cartons and air containers. It allows packing single and mixed size items, optimising the space. For the loading process it considers constraints such as priority loading, stacking rules, load instructions and more.
- 3D LoadPacker [1]: offers a 3D packer that optimises space by compacting distributions of different 3D size rectangular small items within a larger 3D rectangular large object. It considers the total weight of the cargo and considers the orientation of each item type as a constraint.
- LoadMaster [7]: is software that calculates the total volume and gross weight of selected and packed cargo. The user can specify loading rules such as item orientation, stacking limits and maximum overhang, where the cargo on top is not required to be fully supported. Other constraints are load-bearing strength, the fragility of the products.
- LoadPlanner [8]: provides a comprehensive solution for planning and optimising the packaging, palletizing, container and truck loading of cargo. This software is able to handle the complex variety of rules and limitations of the transport industry. It can be used to classify items, orders, containers, etc.
- Cube-IQ [5]: is load planning software that helps estimate the cargo and maximise the volume used. This software can be used to load containers, trucks and rail cars, as well as pallets and cartons. It determines how to arrange the various product types, which can be boxes, rolls, nested L-shapes, nested T-shapes and even trapezoids. It offers flexibility in loading and stacking rules. When it comes to the loading process, it takes into account the loading sequence, weight distributions and orientations, and it stores and retrieves complete loading cases.

Most of the applications studied are lacking the ability to handle controls easily and dynamically. In other cases, the applications are fairly powerful but cannot be used to display the set of solutions derived from the application of the various filling heuristics implemented to solve the multi-objective 3DPP. Because of this, it was necessary to develop and implement a GUI that allowed us to graphically represent the various solutions before applying the different heuristics considered in

Section 3.4. This simplifies the final process for the user of analysing and comparing the solutions.

5.5 3D Graphical User Interface for the 3DPP

We propose the 3DPP-GUI, a Three-Dimensional Graphical User Interface (3DGUI) for the purpose of displaying the process of loading a single container. This interface allows users to see how items (boxes) are to be placed in the container.

Usually, the theoretical studies that solve the 3DPP present their results in a text format. The files that 3DPP-GUI uses are also text files. Since a user will not generally understand the layout of the files that are obtained after applying the different heuristics presented in this paper, it was necessary to develop software that disengages the user from these files. To this end, we implemented a GUI that allows the user, through certain features, to recover the file with the feasible solution and to display the result on the screen allowing the user to alter the container loading process. The 3DPP-GUI designed works with two different files, an input file and a solution file. The input file defines the values and components for the 3DPP, while the solution file has the set of coordinates for each item inside the container. This file is output by the resolution algorithm.

5.5.1 Input File Format

The user must choose the input file containing the definition of the problem. This file is the same file that the evolutionary algorithm will use, which contains information about the pieces and container. Based on this file, the 3DPP-GUI can determine the dimensions (l_i, w_i, h_i), weight (p_i), demand (b_i) and possible orientations $(o_i = 2 * or_1 + 2 * or_2 + 2 * or_3)$ of each type of piece $t_i \in \{t_1, \ldots, t_m\}$. Since the process for creating these files was explained in Section 4.3, only a brief review is provided in this section. The input file has the following format:

L W H *(dimensions of the container)*
P_{max}
m
l_i or_1 w_i or_2 h_i or_3 p_i b_i *(each type of item)*

The container's dimensions are specified in the first row of the file, where L corresponds to the length and W and H to the width and height, respectively. The following data are provided by way of example:

```
587 233 220
3.30986e+07
3
195.667 0 111.064 0 73.3333 1 1.59365e+06 9
195.667 0 60.8763 1 73.3333 1 873508 9
195.667 1 61.0594 1 73.3333 1 876135 9
```

As this shows, the file contains one row for each item type t_i, which specifies its dimensions, weight and demand, along with the types of rotations that can be applied to each item. In the example, the items of type t_1 can only be rotated about their base $o_{t_1} \in [0, 1]$, while the items of type t_2 can rotate $o_{t_2} \in [0, 3]$ and t_3 can rotate freely, $o_{t_3} \in [0, 5]$.

5.5.2 Solution File Format

After applying the different solution heuristics proposed to solve the container loading problem, a file is generated in text format that presents one possible solution for the problem contained in the input file. This files gives the set of Cartesian coordinates (x, y, z) for each item q_j belonging to the problem solution. So as to ascertain the dimensions of each item q_i, this same file also specifies the type of rotation o_i that has been applied and the item type t_i to which each item belongs. With this information, the 3DPP-GUI is able to determine the exact position in which to place each specific item q_i, as well as the type of rotation to apply. The dimensions in the input file are the original ones for each item type t_i. However, when an item q_i is evaluated and placed inside the container as part of the solution, the original dimensions may vary depending on the type of rotation applied to the item. As a result, the solution file format is as follows:

t_i o_i x-axis y-axis z-axis (for each item inside the container)

As we can see, all of the rows that make up the file have the same format. Each row identifies a single item, all of them belonging to the problem solution. Thus, items n_i ($n_i \leq b_i$) that do not belong to the solution are not considered in this file. Below is an example of this based on the example presented for the input file:

```
2 0 0 0 0
2 0 0 0 195.667
2 0 0 73.3333 0
2 0 0 73.3333 195.667
2 0 0 146.667 0
2 0 0 146.667 195.667
2 0 61.0594 0 0
2 0 61.0594 0 195.667
2 0 61.0594 73.3333 0
1 0 61.0594 73.3333 195.667
1 0 61.0594 146.667 0
```

5.5.3 Graphical User Interface

Based on the problem defined and on the specification for the files given in Section 5.5, we have designed a 3D graphical user interface that lets the user graphically display both the problem instances, input file, and the problem solution, solution file. Moreover, and so as to offer an interface with which the user can interface, we have implemented a set of interactive options. These include selecting and loading the files, changing item positions on demand, displaying the elements such as the container and the items from different perspectives, and more. As noted earlier, 3DPP-GUI lets users see the entire loading process based on the set of solutions output after applying the various heuristics.

3DPP-GUI consists of two well differentiated areas. As Figure 5.1 shows, the top part, labelled with the number 1, corresponds to the display and interaction involving the container and the solutions. The bottom part of the image, labelled with the number 2, is used to display the different item types that comprise the input instance of the problem. A description of each element that makes up the interface follows below, the goal being to make it easier for the user to understand the packing process.

- Load the files: when the 3DPP-GUI is run, the user must choose the input file and the solution file. To make this selection, the user must click on the *Choose file or Browse* option, depending on the browser, as shown in Figure 5.2. The format of the files is then verified to be correct. If it is not, a popup is shown to the user informing him of the error. The data contained in the files are processed and are then used by the 3DPP-GUI to arrange the items.
- Draw axes: used to show or hide the coordinate axes for every element in the interface when the box is checked (Figure 5.3).

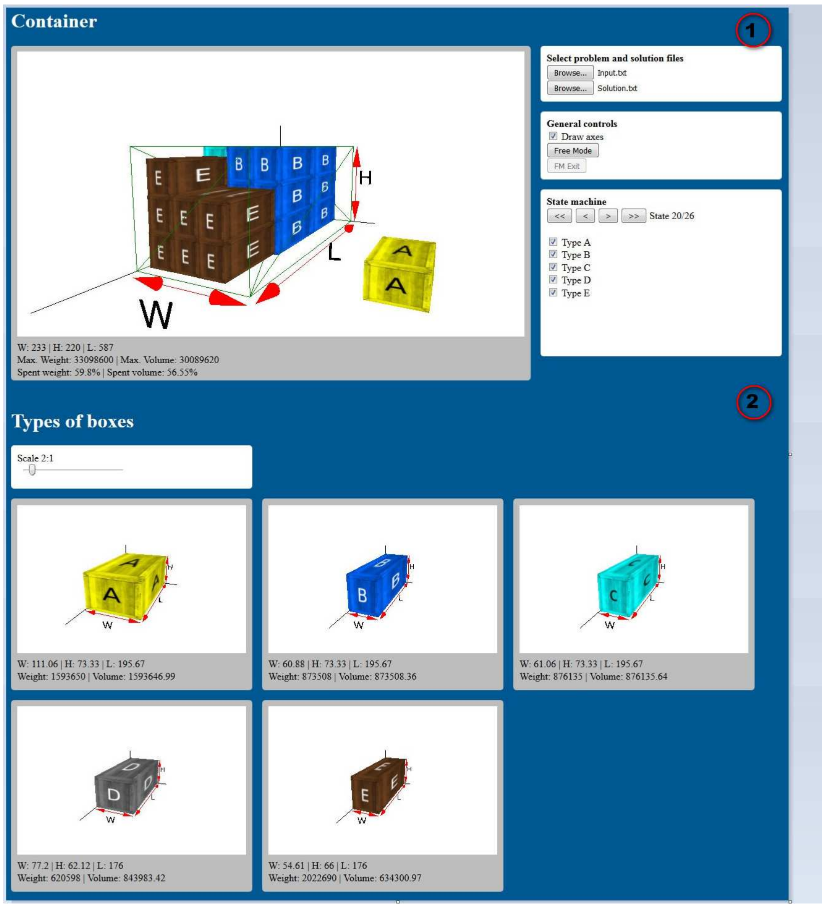


Figure 5.1: 3DPP-GUI work space

- Change scale: zooms the camera closer to or away from the objects. This allows altering the display of the different box types, but not their sizes, which remain the same (Figure 5.4).
- State machine: once the solution with the initial packing pattern is loaded, the user may want to hide certain groups of items to better see the distribution. Two methods are available for doing this: the first is the state machine, where the user can trace the packing process for the items based on the data contained

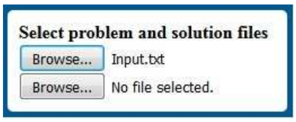


Figure 5.2: Buttons for loading the input and solution files

Figure 5.3: Draw the coordinate axes

Figure 5.4: Scale slider

in the solution file, iteration by iteration, by clicking on the arrow buttons. These buttons let the user step forward or backward through the packing process. The other method implemented can be used to hide different groups of item types by using the checkbox shown in Figure 5.5.

- Free mode: this is a state of the main controller where the user can interact with the items inside the container (Figure 5.6). The user can drag on the various items that make up the solution to different locations inside the display plane (Figure 5.7). This option expands the features of the state machine by allowing the user to manipulate the solutions without having to hide item types or follow a trace to see the arrangement of the contents inside the container. The events that take place are generated due to the user's interaction with the controller. When the user exits free mode, the items return to their original state.

Once the input file is chosen, the graphical system shows the set of available items and a grid which defines the dimensions of the large rectangular object (container). Every item has associated with it a length, width, height, weight and volume, as shown in Figure 5.8. Lastly, when the user selects the solution file, the system draws the arrangement of the packing pattern inside the container, as shown in Figure 5.9. On this same screen, the 3DPP-GUI also shows the values obtained for two objectives of the 3DPP, the volume and weight. The corresponding value is also shown as a percent.

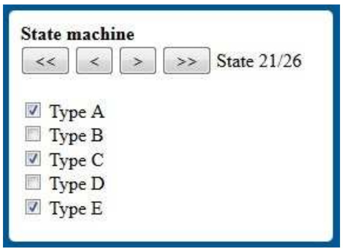


Figure 5.5: State machine

Figure 5.6: Free mode buttons

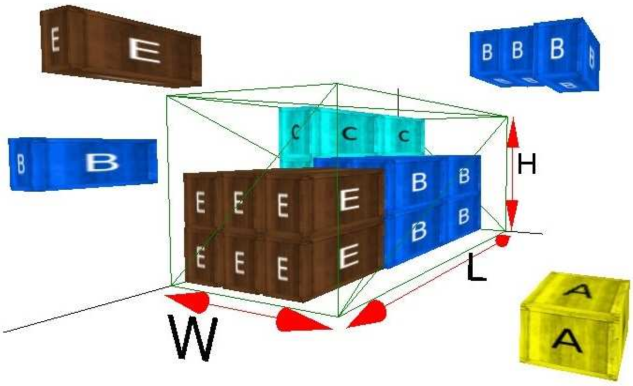


Figure 5.7: Free mode

When the state machine option is used, the values of the objectives are modified based on the button pressed (Figure 5.5). When items are placed inside the container, i.e. when the state machine goes forward, the weight of the cargo increases, as does the volume used. The reverse happens when items are removed from the container. This informs the user of the capacity and weight loaded into the container at all times.

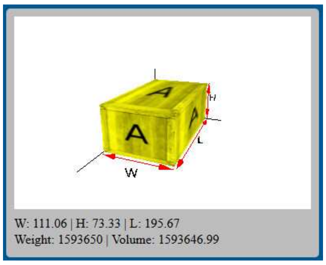


Figure 5.8: Characteristics of the items

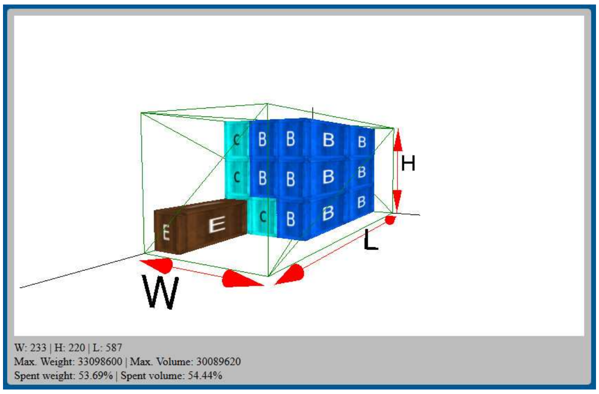


Figure 5.9: Representation of a solution for the multi-objective 3DPP

Chapter 6

Computational Results

In this chapter we present the set of computational studies that were carried out for different sets of instances. So as to apply Multi-Objective Evolutionary Algorithms (MOEAs) to solve the Three-Dimensional Packing Problem (3DPP), we conducted a review of the bibliography on existing meta-heuristics-based frameworks. We also present a set of metrics used for the computational study. The rest of the chapter focuses on presenting the set of studies done to validate the proposals developed over the course of this thesis; namely, we conducted studies to validate the use of Multi-Objective Evolutionary Algorithms (MOEAs) to solve the Three-Dimensional Packing Problem (3DPP), as well as the filling heuristics. Lastly, we carried out a set of experimental evaluations for the purpose of validating the input parameters used by the Three-Dimensional Multi-Objective Generator (3D-MG) to generate the various instances.

6.1 Frameworks

In previous chapters we mentioned that MOEAs offer a promising approach for solving real-world Multi-Objective Optimisation Problem (MOP), yielding good results when used to solve multi-objective problems in the area of cutting and packing [62]. For this reason, the 3D multi-objective packing problem proposed was tackled by applying MOEAs. However, to achieve an efficient and robust optimiser when applying MOEAs to new problems, it is usually essential to test a wide variety of MOEAs and evolutionary operators. Users have a specific view of the problem itself, but they usually only have a general knowledge of MOEAs, their implementation and customisation. Thus, this necessary tuning of a MOEAs when applied to a particular problem, or even a problem instance, hinders the use of these techniques by non-expert users.

In the literature one can find a large number of optimisation tools that support a multitude of techniques: local search algorithms [103], branch-and-bound and general tree searches [15, 140, 208], evolutionary computation [33, 74, 86, 101, 135, 142, 146], etc. Most of these tools are designed to deal with single-objective optimisation problems. In recent decades, however, several specific metaheuristic-based frameworks have been proposed for solving MOPs:

- *PISA* [33]: is an acronym for “A Platform and Programming Language Independent Interface for Search Algorithms”. It is one of the best-known platforms for multi-objective optimisation. The principle underlying PISA is to separate the algorithm-specific part of an optimiser from the application-specific part. Communications between both parts are carried out using a shared-file mechanism, meaning the execution of applications developed using PISA has an associated time penalty.

- *Open BEAGLE* [101]: is a C framework that provides a high-level software environment to do any kind of evolutionary computation. It supports tree-based genetic programming; bit string, integer-valued vector, and real-valued vector genetic algorithms; and evolutionary strategies.

- *jMetal* [74]: is an acronym for “Metaheuristic Algorithms in Java”. It is an object-oriented Java-based framework aimed at the development, experimentation, and study of metaheuristics for solving multi-objective optimisation problems. It provides a rich set of classes that can be used as the building blocks of multi-objective metaheuristics, taking advantage of code-reusing, thus facilitating the development of new multi-objective algorithms.

- *ParadisEO* [146]: is a template-based, ANSI-C++ compliant framework dedicated to the flexible design of metaheuristics, including solution-based metaheuristics (local search, simulated annealing, iterated local search, etc.) and population-based metaheuristics (genetic algorithm, particle swarm optimisation, evolutionary strategy, etc.). It also provides tools to design metaheuristics for multi-objective optimisation.

- *METCO* [142]: is an acronym for “Metaheuristics-based Extensible Tool for Cooperative Optimisation”. It is a framework that supports the implementation and execution of sequential and parallel meta-heuristics. Its functionality can be expanded by defining new plug-ins. This software was designed so that users can adapt the environment to the requirements of their problem, and to adapt the configuration of the MOEAs.

Despite the large number of tools, each has its advantages over the others. For example, PISA [33] incorporates a large number of metrics that make it easy to carry out comparisons, ParadisEO [146] support various parallel models, jMetal [74] features the most optimisation algorithms, and METCO [142] allows for a great deal of customisation of the island-based parallel models.

The definition of the 3DPP considered in this study, as well as the implementation of the various heuristics, were carried out using the METCO tool. METCO relies on evolutionary techniques to solve a MOP. It allows carrying out sequential executions with various evolutionary algorithms and integrating these techniques into collaborative parallel models. The METCO user interface allows configuring the framework by specifying a set of plug-ins. Through these plug-ins, new algorithms can be introduced, the problem to be solved can be specified, and practically any characteristic of the sequential and parallel models to be executed can be modified. The plug-in is developed by specifying a set of classes in C++ inherited from a class developed in METCO. Although METCO features a selection of the most common MOEAs in the literature, others can be added by using plug-ins [143, 145]. By customising the configuration files and developing new plug-ins, users can adapt the framework to their own requirements. The software has been designed in such a way that the user can easily specify the problem requirements and customise the configuration of the MOEAs that will be involved in the problem solution. METCO contributes with a variety of libraries that provide a set of standard evolutionary computation operations, such as mutation, crossover and selection operators.

METCO provides a parallel model based on dynamic island allocation. It is intended to offer an easily customisable island-based model that includes mechanisms to adapt the behaviour of the islands over the course of the executions [144]. The scheme does this by trying to detect and allocate more resources to the most suitable algorithms and operators. This adaptive model is a hybrid algorithm that combines a parallel island-based scheme with a hyperheuristic approach [141]. The tool allows for extensive customisation of the island-based models so that expert users can adapt it to obtain even better optimisers.

Once the required plug-ins are implemented, the framework combines them, allowing the tool to be used in parallel and sequential mode. For the sequential execution, the user need only specify the problem name, the stopping criterion and the algorithm to be used, along with its corresponding parameters (population size, mutation and/or crossover operators, etc.). In the case of parallel executions, the user must specify each MOEA used in the model along with every parameter for the parallel model utilised: migration mechanism, stopping criteria, parameters for the scoring/selection strategies, etc.

6.2 Multi-Objective Metrics

The literature features a large variety of performance metrics or indicators that can be used to analyse results. The metrics compare and quantify the quality of the resulting Pareto front, taking into account aspects such as the distance to the true front, and the amplitude and distribution of the solutions on the front. There is currently no universal metric, since each measure proposed offers a set of advantages and drawbacks. A brief review of the literature takes us to 1994, when Srinivas and Deb [201] used the Chi-squared distribution to analyse how well the individuals obtained by the NSGA and the VEGA are distributed along the non-dominated region. In 1996, Fonseca and Fleming proposed a statistical method based on the concept of (attainment surfaces) [98, 136]. The hypervolume metric formulated by Zitzler and Thiele [231] was developed in 1998. The so-called C-metric (Set Coverage), also introduced by Zitzler and Thiele [231] in the same year, compares the quality of two fronts. This coverage metric $C(A, B)$ calculates the percentage of solutions in B that are weakly dominated by at least one of the solutions of A. In 1999, Van Veldhuizen [212] proposed several metrics: Error ratio (ER), which measures what percentage of the current Pareto front does not belong to the true Pareto front; generational distance (GD), which estimates how distant the solutions of the calculated front are from the true front; Inverse generational distance (IGD), which represents the average distance of the solutions in the current front to the true Pareto front; and Spacing (SP), which evaluates the uniformity of the Pareto front distribution along the space it occupies. The M1, M2 and M3 metrics arose in the year 2000 [227] and were used to measure the average distance to the Pareto front, the distribution of the non-dominated solutions and the diversity.

The best way to efficiently evaluate the performance of a multi-objective algorithm is to employ different metrics. The hypervolume (HV) [231] is one of the metrics used in the experiments presented in the next section. It is defined as the volume enclosed by the front; that is, it measures the region dominated by the front. In order to carry out this measurement, a dominated reference point must by chosen, which will serve as the limit of the enclosed area so the volume can be calculated. The value of this metric should be large, since these means that the points are spread out in the solution space. Another of the metrics employed is the attainment surfaces (AS) [98], which yields a given probability that an area of the space is dominated. These metrics offer a simple way to view the results in a given number of executions.

6.3 Validation of the Sequential Algorithms

The purpose of this computational study is to solve the Multi-Objective 3DPP presented in Section 3.4 using evolutionary strategies offered by METCO [142]. The Multi-Objective Evolutionary Algorithms (MOEAs) have provided competitive results when applied to some cutting and packing problems. Thus the goal now is to check their effectiveness for the multi-objective 3DPP. This work makes its contribution by providing a multi-objective solution to a problem like the 3DPP, introducing weight as the second objective. This problem has always been formulated as a mono-objective problem in the literature. In an effort to validate our multi-objective proposal, we compared the results obtained when using the various filling heuristics applied to different MOEAs against the known results in the literature presented in Dereli et. al [70], which consider the problem with multiple objectives and that also take into account the same objectives as in this paper. In the referenced work [70], the problem is solved using a Simulated Annealing (SA) algorithm with a filling heuristic similar to the Single-Level Filling Heuristic in Depth (SLFHD) approach implemented in this work. The heuristic proposed in [70] is a kind of wall-building method in which the container is filled layer by layer. The parameters of the simulated annealing algorithm used in [70] for the 3DPP are: initial temperature $T_{in} = 5000$, final temperature $T_f = 0.0001$, the constant $\alpha = 0.987$ and number of iterations to be performed for each temperature $il_{max} = N$ (equal to the number of box types in each instance problem). They also assigned different weights, from 0 to 1 by 0.1 increments/decrements, to each objective, turning the multi-objective problem into a mono-objective problem, with the multiple objectives being combined in a single evaluation function. However, and for the purpose of being able to make a comparison, the work presented by Dereli et. al [70] will be used as a reference to check the competitiveness of our proposals using different MOEAs, such as Non-Dominated Sorting Genetic Algorithm II (NSGA-II), Strength Pareto Evolutionary Algorithm 2 (SPEA2) and Adaptive Indicator-Based Evolutionary Algorithm (Adaptive-IBEA). It is thus interesting to study the performance of the filling heuristics presented in Section 3.4.3, as we are concerned with a formulation that is actually multi-objective in which the goal is to maximise not only the volume used, but also to consider the weight. Once we validate that the formulation applied is suitable for solving the 3DPP, we will then study the performance of the different filling heuristics presented in Section 3.4.3.

For the computational study, we have used the real test problem instance proposed by Dereli et al. in [70]. It involves an order for 12 different product types distributed by the company *Procter & Gamble*. The company distribute their own products (paper, towels, toilet tissues, paper napkins, etc.) in their own transport

ID	Description	w/l/h (cm)	No. of boxes	Weight (kg)	Total Volume (cm^3)
1	Detergent	40/36/28	325	20.00	13,104,000
2	Bleaching liquid	54/28/30	25	22.00	1,134,000
3	Personal care	54/28/30	75	2.35	3,402,000
4	Detergent	39/29/32	75	20.00	2,714,400
5	Shaving product	15/10/20	10	1.47	30,000
6	Baby care	42/37/25	150	2.80	5,827,500
7	Toothpaste	36/18/18	3	3.72	34,992
8	Shampoo	18/17/22	25	4.97	168,300
9	Shampoo	22/17/22	50	5.00	411,400
10	Shampoo	12/11/16	5	1.33	10,560
11	Bleaching liquid	30/27/40	20	17.60	648,000
12	Shaving product	19/7/21	3	0.12	8,379
Total			766	9905.37	27,493,531

Table 6.1: Actual test-problem instance

vehicles. The quantities, dimensions ($Length \times Width \times Heigth$) and weights are specified for both the vehicle and the products. The vehicles have a capacity of $(530 \times 220 \times 210)cm^3$ and can hold a maximum weight of $7200kg$. The instance has 12 types of products with different associated sizes and weights (Table 6.1); however, there is no information on the boxes' orientations. Therefore, and because some of the boxes contain liquids, it is assumed that the boxes can only be rotated about their base.

6.3.1 Description of the Experiment

The set of experimental evaluations included in this section were run on a dedicated Debian GNU/Linux operating system with 48 cores, each consisting of one *AMD Opteron 6164HE*. The METCO framework and the problem approach were implemented using C++ and compiled with *gcc 4.7.2*. In this study, for the experimental evaluation, we checked the behaviour of three of the best-known MOEAs, NSGA-II, SPEA2, and an adaptive version of IBEA, which were applied to the different filling heuristics: SLFHL, SLFHD, MLFHL, and MLFHD. We checked the algorithms with different population sizes $(20, 25, 50, 100)$, mutation rates $(0.1, 0.2, 0.3)$ and crossover rates $(0.7, 0.8, 0.9)$. Since we are dealing with stochastic algorithms, each execution was repeated 30 times. Table 6.2 shows the configurations used for the MOEAs employed in the study. The fitness scaling factor, which was set to 0.001, is a parameter that is exclusive to the IBEA and the Adaptive-IBEA. The average

MOEAs parameter	Tuning values
Population sizes	**20**, 25, 50 and 100
Mutation rates	0.1, 0, 2 and **0.3**
Crossover rates	0.7, 0.8 and **0.9**
Fitness scaling factor	0.001
Evaluations	**30000**
Generations	**1500**

Table 6.2: MOEAs - Initial configuration

time used for each execution was around 10 minutes.

6.3.2 Analysis of the Experiment

During the process of developing the different variation operators, they were validated for the purpose of determining if they improved the results. The mutation operators studied were the add one gene mutation, the remove one gene mutation, the change a gene mutation, the modify some genes mutation, the swap two genes mutation, and the shift a gene mutation. Of these, the combination of the add one gene mutation, the remove one gene mutation, and the change a gene mutation was found to yield good results, whereas adding mutation operators such as the modify some genes mutation, the swap two genes mutation, and the shift a gene mutation yielded no improvements. In some cases, they even worsened the solutions. Thus, for the group of experiments carried out in this chapter, we used the add one gene, remove one gene, and change one gene mutation operators. Each of which applied under the probability of mutation p_m.

With this first study, our goal is to demonstrate that it is possible to obtain good quality solutions by using truly multi-objective formulations. An additional goal is to validate the good performance of MOEAs when solving complex cutting and packing problems like the 3DPP. From each of the 30 executions carried out for each of the MOEAs analysed, along with the filling heuristics, the final solution set is analysed in order to select the highest average volume (Sol.Volume) and the highest average weight (Sol.Weight). Table 6.3, Table 6.4 and Table 6.5 show the solution with the average-highest volume and the solution with the average-highest weight when the various filling heuristics are applied to NSGA-II, SPEA2, and Adaptive-IBEA, respectively. These values are calculated by finding the general average of the best volumes and weights versus the final front for each of the executions carried out. Thus, Sol.Volume corresponds to the best volume obtained when executing a given

Objective	SLFHL		SLFHD		MLFHL		MLFHD	
	Sol. Volume	Sol. Weight	Sol. Volume	Sol. Weight	Sol. Volume	Sol. Weight	Sol. Volume	Sol. Weight
Volume (%)	91.21%	91.48%	89.49%	90.06%	91.42%	91.20%	89.61%	90.12%
Weight (kg)	7199.64	7195.64	7199.43	7195.03	7199.95	7197.94	7198.63	7191.58

Table 6.3: Comparison of objectives for NSGA-II: volume and weight

Objective	SLFHL		SLFHD		MLFHL		MLFHD	
	Sol. Volume	Sol. Weight	Sol. Volume	Sol. Weight	Sol. Volume	Sol. Weight	Sol. Volume	Sol. Weight
Volume (%)	91.15%	91.42%	89.32%	88.35%	91.33%	91.16%	89.96%	89.86%
Weight (kg)	7197.92	7198.03	7192.34	7194.71	7198.84	7197.57	7199.95	7189.51

Table 6.4: Comparison of objectives for SPEA2: volume and weight

Objective	SLFHL		SLFHD		MLFHL		MLFHD	
	Sol. Volume	Sol. Weight	Sol. Volume	Sol. Weight	Sol. Volume	Sol. Weight	Sol. Volume	Sol. Weight
Volume (%)	90.93%	89.34%	88.28%	85.80%	90.97%	91.29%	89.41%	90.95%
Weight (kg)	7199.64	7196.94	7199.79	7187.67	7198.92	7196.83	7198.37	7195.84

Table 6.5: Comparison of objectives for Adaptive-IBEA: volume and weight

configuration. The same applies to Sol.Weight, and its value is calculated based on the best weights obtained by the last 30 Pareto fronts. Finally, and in an effort to do a comparison with the only valid results found for dealing with multi-objective problems, the four solutions obtained by the filling heuristic presented by Dereli et al. in [70] are included in the Table 6.6.

In Table 6.3, Table 6.4 and Table 6.5, the average highest volume is expressed as a percent of the container volume used. For all cases, the value of the other objective is also shown. This allows us to see the quality of both objectives for the best average based on one of the objectives. From the tables we can see how when the SLFHL approach is used for the NSGA-II (91.21%), the results are improved when the same filling heuristic is executed with the SPEA2 (91.15%), and the 90.93% obtained by Adaptive-IBEA. And hence all of these results improve on the values in

Objective	Dereli and Sena 2010			
	Sol. 1	Sol. 2	Sol. 3	Sol. 4
Volume (%)	87.51%	86.13%	86.09%	85.96%
Weight (kg)	6713.37	7157.06	7150.41	7151.37

Table 6.6: Comparison of objectives for reference work: volume and weight

the reference paper shown in Table 6.6. When we study the values yielded when the SLFHD heuristic is used, the average highest volume is achieved using the SPEA2 algorithm (91.15%), followed by NSGA-II (89.49%) and Adaptive-IBEA (88.28%). However, every case manages to improve on the results obtained when using the filling heuristic with the Simulated Annealing presented in [70]. When we analyse the average values obtained when using filling heuristics with multiple-levels, they all improve on the values obtained when using filling heuristics with a single-level, as well as on the results obtained by Dereli et al. [70]. Thus, when the analysis is conducted based on MLFHL, we see how the best values are obtained when NSGA-II is used (91.42%), versus the 91.33% yielded by SPEA2 and the 90.97% by Adaptive-IBEA. Finally, when the MLFHD heuristic is used, we see how it is SPEA2 that yields the best average volume (89.86%), followed by NSGA-II, with a percentage of 89.69%, and Adaptive-IBEA with 89.41%. This testing verifies the good behaviour of SPEA2 when used with the MLFHD heuristic, as well as the good performance of NSGA-II when executed with the MLFHL and both single-level approaches (SLFHL, SLFHD).

A similar process is repeated for the weight objective. If we calculate the average of the highest weight in each of the 30 final fronts using the NSGA-II, SPEA2 and Adaptive-IBEA algorithms with the SLFHL heuristic, we obtain a value of 7195.64kg, 7198.03kg and 7196.94kg respectively for each of the algorithms. At this point, we can see that the best average weight when using SLFHL is obtained by SPEA2, though unlike with the other two algorithms, it is not very significant. In the case of SLFHD, we see more of a difference between the values obtained, these being 7195.03kg, 7194.71kg and 7187.67kg for NSGA-II, SPEA2 and Adaptive-IBEA. The highest average weight is yielded by NSGA-II. The results, thus far evaluated for weight, improve on the four results presented in the reference paper by Dereli et al. We can thus confirm that the multi-objective approach allows us to improve on the only results that exist for the 3DPP that deal with more than one objective. If we now consider the values obtained when applying the MLFHL heuristic, we see that the values output by NSGA-II (7197.94kg) improve on those obtained by SPEA2 (7197.57kg) and Adaptive-IBEA (7196.83kg). However, when we study the

MLFHD, the best average weight is given by Adaptive-IBEA at $7195kg$, versus the $7191.58kg$ and $7189.51kg$ output by NSGA-II and SPEA2. As in the previous case, the best average weights are found by the heuristics that work with more than one level (MLFHL, MLFHD) versus the single-filling heuristics.

From this initial study, we may conclude that formulating the problem directly as multi-objective improves on the only known results that consider the same set of objectives. Thus, including the weight as a second objective guarantees good results not only in terms of volume, but of weight as well. This multi-objective approach also provides a more direct parallel to the real-world industry, where weight is an important factor to consider when loading containers.

Having validated the effectiveness of applying multi-objective techniques to solve the 3DPP, we now proceed to check the performance of the three MOEAs used: NSGA-II, SPEA2 and Adaptive-IBEA. In order to provide the results with confidence, comparisons were carried out as per the following statistical analysis [69, 195]. First, a Kolmogorov-Smirnov test is performed in order to check whether the values of the results follow a normal (Gaussian) distribution or not. If so, the Levene test checks for the homogeneity of the variances. If samples have equal variance, an ANOVA test is done; otherwise a Welch test is performed. For non-Gaussian distributions, the non-parametric Kruskal-Wallis test is used to compare the medians of the algorithms. A confidence level of 95% is considered, which means that the differences are unlikely to have occurred by chance with a probability of 95%. The analysis is performed using the hypervolume [231] metric and attainment surfaces [98]. Table 6.7 and Table 6.8 show the statistical comparison between the different algorithms studied when applying the single-level and the multiple-level filling heuristics in length to the different MOEAs, using the parameters that perform the best after the initial tuning (Table 6.2). Table 6.9 and Table 6.10 show the same comparison, but using the corresponding filling heuristics in depth. The symbol $\uparrow$ is used to denote that differences between the models are statistically significant and that the model in the left column yields a higher median and mean value. In the cases in which the opposite occurs, the symbol $\downarrow$ is used. Finally, for the cases in which the differences were not statistically significant, the symbol $\leftrightarrow$ is used. NSGA-II and SPEA2 yielded statistically better results than Adaptive-IBEA. Therefore, the Adaptive-IBEA exhibited the worst performance, regardless of the filling heuristic used to evaluate the solutions. When evaluating the statistical performance of the NSGA-II and SPEA2 algorithms with the different types of filling heuristics, we see that in some cases one performs better, statistically, than the other, though both algorithms exhibit good performance.

MOEAS	SLFHL		
	NSGA-II	SPEA2	Adapt-IBEA
NSGA-II SLFHL	↔	↔	↑
NSGA-II MLFHL	↑	↑	↑
SPEA2 SLFHL	↔	↔	↔
SPEA2 MLFHL	↔	↔	↑
Adapt-IBEA SLFHL	↓	↔	↔
Adapt-IBEA MLFHL	↔	↔	↔

Table 6.7: Statistical comparison among MOEAs when using SLFHL

MOEAS	MLFHL		
	NSGA-II	SPEA2	Adapt-IBEA
NSGA-II SLFHL	↓	↔	↔
NSGA-II MLFHL	↔	↔	↑
SPEA2 SLFHL	↓	↔	↔
SPEA2 MLFHL	↔	↔	↑
Adapt-IBEA SLFHL	↓	↓	↔
Adapt-IBEA MLFHL	↓	↓	↔

Table 6.8: Statistical comparison among MOEAs when using MLFHL

MOEAS	SLFHD		
	NSGA-II	SPEA2	Adapt-IBEA
NSGA-II SLFHD	↔	↔	↑
NSGA-II MLFHD	↔	↔	↑
SPEA2 SLFHD	↔	↔	↑
SPEA2 MLFHD	↑	↑	↑
Adapt-IBEA SLFHD	↓	↓	↔
Adapt-IBEA MLFHD	↔	↔	↑

Table 6.9: Statistical comparison among MOEAs when using SLFHD

MOEAS	MLFHD		
	NSGA-II	SPEA2	Adapt-IBEA
NSGA-II SLFHD	↔	↓	↔
NSGA-II MLFHD	↔	↓	↑
SPEA2 SLFHD	↔	↓	↔
SPEA2 MLFHD	↑	↔	↑
Adapt-IBEA SLFHD	↓	↓	↓
Adapt-IBEA MLFHD	↓	↓	↔

Table 6.10: Statistical comparison among MOEAs when using MLFHD

In the next study conducted, we compared the performance of the single-level filling heuristics and the multi-level filling heuristics in an effort to determine whether using a certain order when filling the container improves the results or not. Table 6.8 and Table 6.10 show that, in fact, the order in which the boxes are placed inside the container has an important effect that must be considered when applying the filling

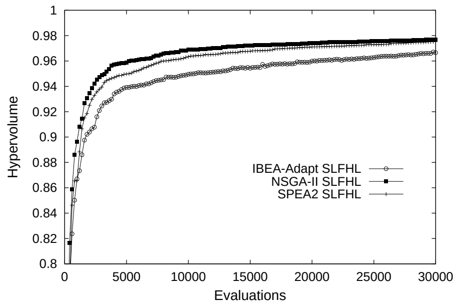
MOEAS	SLFHL			MLFHL		
	NSGA-II	SPEA2	Adapt-IBEA	NSGA-II	SPEA2	Adapt-IBEA
NSGA-II SLFHD	↓	↓	↓	↓	↓	↓
NSGA-II MLFHD	↓	↓	↓	↓	↓	↓
SPEA2 SLFHD	↓	↓	↓	↓	↓	↓
SPEA2 MLFHD	↓	↓	↓	↓	↓	↓
Adapt-IBEA SLFHD	↓	↓	↓	↓	↓	↓
Adapt-IBEA MLFHD	↓	↓	↓	↓	↓	↓

Table 6.11: Statistical comparison among filling heuristics in length and in depth

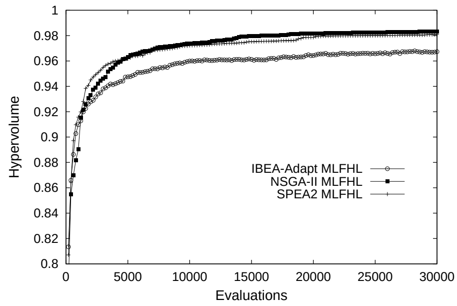
heuristic. This thus verifies that the coding used to code the gene is important, since in our case this coding was done by considering the order in which the boxes are placed inside the container. We thus find that the multi-level filling heuristics exhibit statistically better performance than the single-level filling heuristics. This behaviour is repeated both when applying the filling heuristics using the NSGA-II algorithm and when SPEA2 is used. Therefore, having all of the levels available offers an improvement over those filling heuristics that only allow having one level open during the container loading process.

Finally, we studied the performance of the various filling heuristics when they are used to load boxes into the container, comparing the filling heuristics in length, SLFHL and MLFHL, with the filling heuristics in depth, SLFHD and MLFHD. Table 6.11 shows a statistical comparison when different types of processes are used to pack the container. This study shows that the filling heuristics in length unquestionably yield better statistical results for this real-world problem (Table 6.1). Therefore, after this study we can ascertain that the best performance when using any of the various filling heuristics is obtained with the NSGA-II and SPEA2 algorithms. As for the types of filling heuristics, we have shown that using a certain order for loading boxes into the container is necessary, considering the type of coding selected to code the population. Finally, we conclude that for the instance studied, the best filling heuristics are those that work over the length of the container.

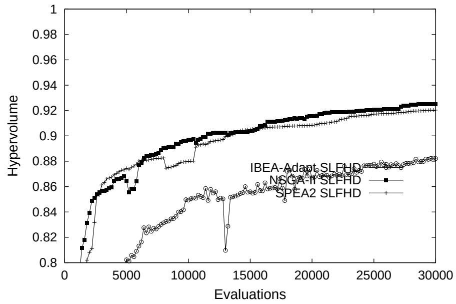
The hypervolume [231] yielded by each algorithm is shown in Figure 6.1. For the analysed instance, we can see that NSGA-II yields quality results when used with any of the four filling heuristics explained in Section 3.4.3. However, when the multi-level filling heuristic in depth is used, this performance is reversed, since the best results are obtained by SPEA2. In addition, as the various graphs show, the worst performance is given by Adaptive-IBEA, the results of which are far worse than those output by NSGA-II and SPEA2. This evaluation thus shows that both NSGA-II and SPEA2 have a very similar behaviour, whereas the hypervolume for the adaptive version of IBEA is slightly lower.



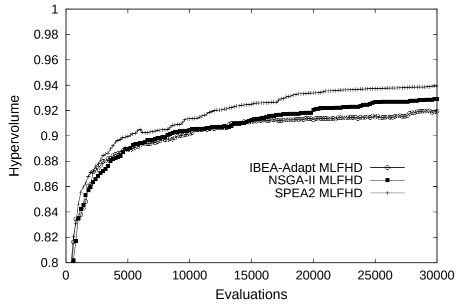
(a) SLFHL



(b) MLFHL



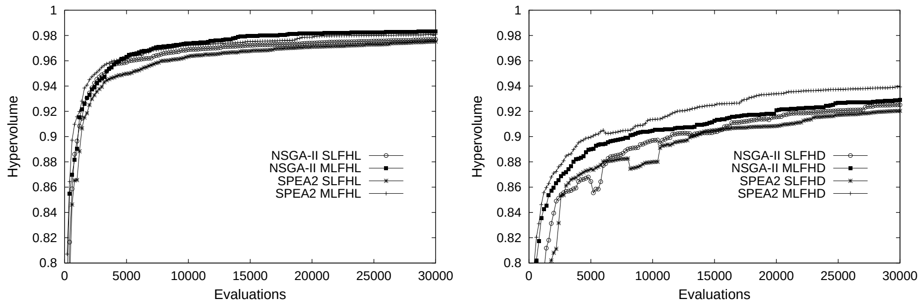
(c) SLFHD



(d) MLFHD

Figure 6.1: MOEAs applied to filling heuristics

Given that the NSGA-II and the SPEA2 obtained the best results among the MOEAs tested, the next experiments check the behaviour using the four filling heuristics presented in this work for both MOEAs. Figure 6.2 shows the hypervolume for NSGA-II and for SPEA2 with every heuristic. If we observe the behaviour of the filling heuristics when the cuts are made along the length of the container, that is, when the cuts are parallel to the y-axis, we can see how both MOEAs yield results of more or less equal quality using the different filling heuristics. However, when we study the behaviour of NSGA-II and the SPEA2 with the filling heuristics in depth (SLFHD, MLFHD), we clearly see that SPEA2 is better when using the MLFHD heuristic. This confirms what was noted in the previous study, since we can conclude that the behaviour of NSGA-II and SPEA2 depends on the heuristic. If we now focus on studying the results output by the single-level filling heuristics and the multiple-level filling heuristics, we clearly see that the results are worse



(a) SLFHL/MLFHL (b) SLFHD/MLFHD

Figure 6.2: NSGA-II and SPEA2 applied to four filling heuristics

with the single-level heuristics, both in length and in depth, with both MOEAs. We may thus conclude that the good behaviour of NSGA-II and SPEA2 depends on the filling heuristic. In general, NSGA-II and SPEA2, using the MLFHL and MLFHD heuristic, show the expected behaviour, with their hypervolume outperforming the best model of the SLFHL and SLFHD filling heuristics. This shows the benefits of filling the container by considering a certain order in the gene.

After concluding in the previous study that the best results were obtained by the multi-level filling heuristics, we conducted the ensuing study in an effort to determine which of the two multi-level filling heuristics, MLFHL and MLFHD, gives the higher quality results. Figure 6.3 shows the results obtained when the filling heuristics are applied making use of the multi-level versions in length and in depth. We can see how the multi-level filling heuristic in depth (MLFHD) does not provide the expected results, which are well below those obtained by the MLFHL heuristic, as shown in Figure 6.4. In addition, Figure 6.4 shows two atypical values, which could be the result of a measurement or recording error. When the multiple-level filling heuristics are executed with NSGA-II and SPEA2, we see that the part containing the $Q1$ and $Q2$ quartiles is larger in every case than that containing the $Q2$ and $Q3$ quartiles. This indicates that the results between 25% and 50% are more scattered than those between 50% and 75%. The median in all of the results appears to be biased toward the $Q3$, indicating that many of the results tend to cluster toward one point on the scale. The MLFHD has the most scattered data distribution, as evidenced by the length of its whiskers. We also see that the MLFHL yields average values that are much higher than those output by the MLFHD for the real instance studied. This could be because when the container is filled using the multi-level

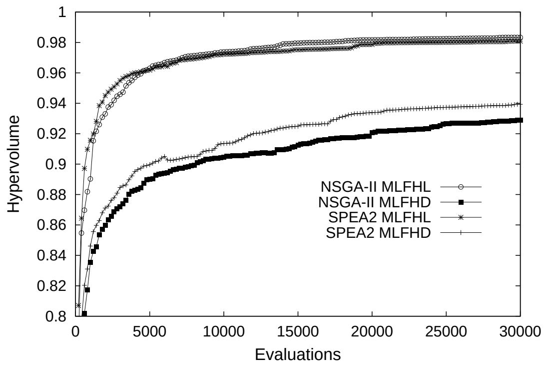


Figure 6.3: NSGA-II and SPEA2 applied to multi-level filling heuristics

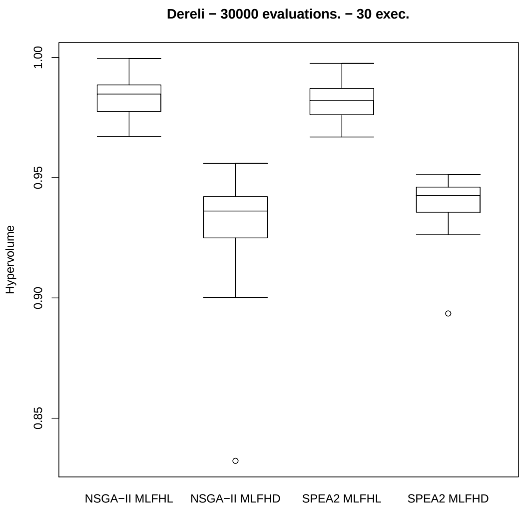


Figure 6.4: Box plot of the multi-level filling heuristics

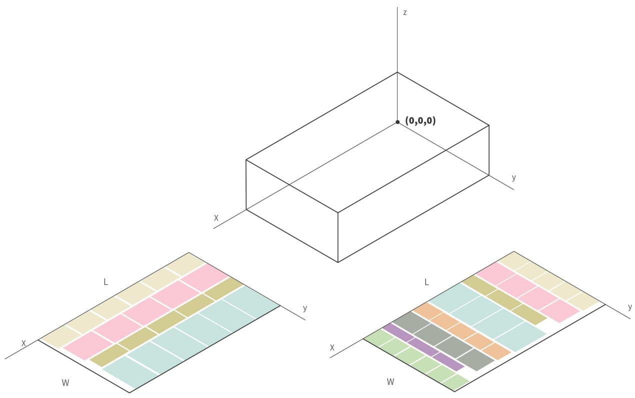
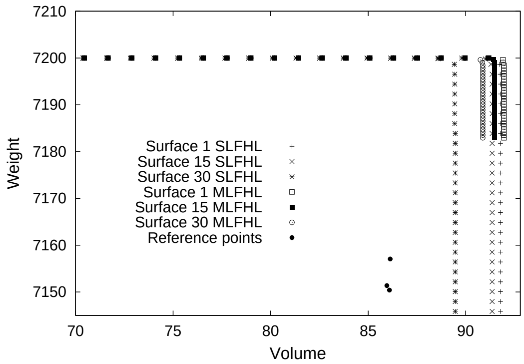


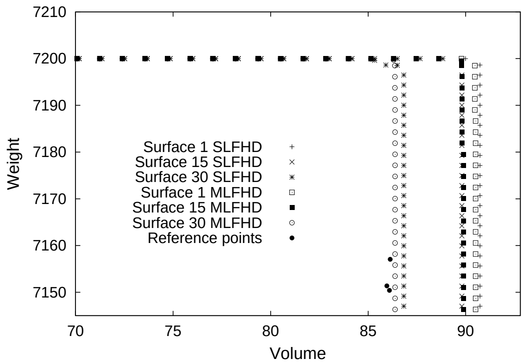
Figure 6.5: Holes when using filling heuristics in length and in depth

filling heuristic in length, less space is wasted than if the multi-filling heuristic in depth is used. As Figure 6.5 shows, when the filling heuristic in length is used, the probability of leaving gaps is lower than if the multi-level filling heuristic in depth is used.

So far we have seen the average of the extreme values for the objectives. But it would be useful to identify the area of the search space being explored and see the general behaviour of the evolutionary algorithm applied to the 3DPP. Identifying the area of the search space being explored and seeing the general behaviour of the evolutionary algorithms applied to this problem could be very confusing if we were to draw the 30 Pareto fronts, so instead we have used attainment surfaces [98]. Figure 6.6 shows 1, 15 and 30 attainment surfaces for the SLFHL, MLFHL, SLFHD and MLFHD heuristics when they are executed using the NSGA-II and SPEA2 algorithms and the solutions reached by Dereli et al. (Table 6.6). As we can see, the results cover a large area of the solution space, offering a set of compromise solutions between the two objectives, such that the decision maker can choose the most appropriate option in each case. If a solution is needed that yields a volume as large as possible, said solution will have a lower weight associated with it than a solution that yields a lower packing volume, and vice versa. However, different solutions given in [70] (reference points in Figure 6.6) are behind attainment surfaces, being completely dominated by the single-level filling heuristics and multi-level filling heuristics, both in length and in depth. The multi-level filling heuristics gave many solutions that are non-dominated by the solutions given by the single-level filling heuristics in length and in depth. Moreover, MLHFL provides better solutions than



(a) NSGA-II



(b) SPEA2

Figure 6.6: Attainment surfaces and existing solutions

those obtained by the MLFHD approach. We may conclude, then, that the MLFHL approach provides high-quality solutions, thus proving the validity of MOEAs when tackling real-world multi-objective problems. And it also proves the validity of the designed heuristics. The first solution $(21427698, 6713.37)$ given by Dereli et al. (Table 6.6) is not shown because it is well below the graph. This is because the weight value is much lower than the solution fronts represented here. But it is dominated too. That way, the validity of MOEAs to tackle such problems is demonstrated, beating the results obtained using a simulated annealing algorithm. Moreover, the set of compromise solutions is obtained in only one execution, whereas the algorithm applied in [70] assigns different weights to each objective, and each weight combination requires an execution.

It should be noted that the way in which the attainment surfaces is presented could be strange, because it seems there are many equal points for the weight and volume, but it only appears like that because of the scales of the axes in the graph. For example, there seem to be many solutions with weight $7200kg$, but if the scale is increased, we see that the values are different, though very similar: $7199.99, 7199.95, 7199.50$, etc.

Based on this initial study, and after analysing several MOEAs, we can conclude that NSGA-II and SPEA2 exhibit good performance for the 3DPP and the instance in question. These algorithms yielded a set of solutions that compromise between the two objectives without requiring several executions with different weights for each objective, as happens with the Simulated Annealing algorithm used as reference. Moreover, said solutions completely dominate the reference results (Table 6.6), which once again demonstrates the good behaviour of the evolutionary strategy with cutting and packing problems.

6.4 Validation of Filling Heuristics in Parallel

Having verified that using MOEAs yields quality results, it would be interesting to see if the island-based parallel models offered by Metaheuristics-based Extensible Tool for Cooperative Optimisation (METCO) can improve on the performance of the sequential models. For the study we selected one of the constructive heuristics that gave bad quality results for the only real instance known to deal with the multi-objective version of the MOEAs so as to test if using a parallel approach can improve the results. Thus, for this parallel approach, the study was based on the Single-Level Filling Heuristic in Length (SLFHL), which provided lower quality results than those obtained by both multi-level filling heuristics.

We conducted two studies involving the same filling heuristics, one using a homogeneous island-based model and another using a heterogeneous island model. The simplest model that can be tested is the homogeneous island-based model, where every island executes the same configuration of a MOEA. In the heterogeneous island-based models, each island executes a different MOEA.

6.4.1 Description of the Experiments

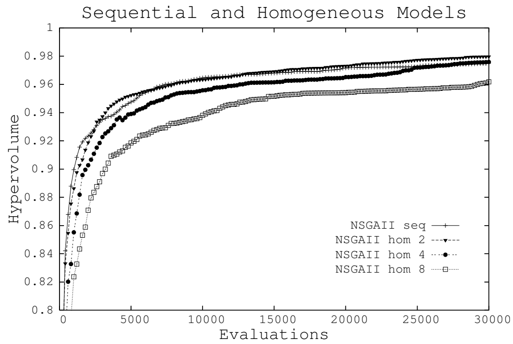
In this case, the experimental evaluation was carried out using a dedicated cluster with 20 dual-core nodes and a Debian GNU/Linux operating system. Both the

METCO framework and the problem approach were implemented using C++ and compiled with *gcc 4.1.3* and MPICH *1.2.7*. The studies were checked using the MOEAs that yielded the best results in the sequential study, NSGA-II and SPEA2. For the computational study, we used the same instance employed in Section 6.3, defined in Table 6.1. The NSGA-II and SPEA2 algorithms were executed in parallel using the same parameters applied in the sequential executions. The migrations take place in each generation with a probability of 0.1, and when carried out, 4 individuals are migrated.

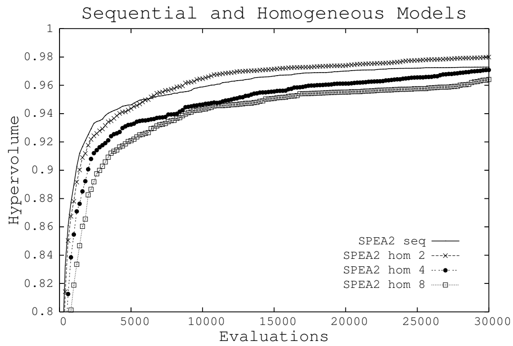
6.4.2 Analysis of the Experiments

First, the homogeneous island-based models are analysed, where every island executes the same configuration of a MOEA. We carried out the executions using the homogeneous model with the NSGA-II algorithm with two, four and eight islands. The same process was performed using the SPEA2 algorithm. Figure 6.7 shows the hypervolume for each version using different MOEAs. The results for a parallel execution with two islands show better performance than the sequential ones, especially when the number of evaluations is increased, and with it, the time. A four-island execution only manages to reach the quality of the sequential ones at the end, while eight islands are not able to improve the quality of sequential solutions in the same total number of evaluations. This is because the more islands that are used, the lower the number of evaluations or generations that each island explores separately, i.e., the same number of evaluations is divided among more processors, and thus each processor makes less headway in the exploration. If we wanted to execute the parallel model during the same time as the sequential one, improvements over the sequential solutions could be possible. However, here we want to execute the same number of evaluations, so the parallel execution times are considerably reduced when compared with the sequential ones.

Thus, if we conduct an in-depth analysis of how many evaluations on average each processor needs to reach a hypervolume percentage (Table 6.12), we see that each processor in the parallel executions carries out fewer evaluations than the processor in the sequential execution. Moreover, we see that when the quality of the solution is increased, the number of evaluations required by each processor increases proportionately. Since not much overhead is required to synchronise the parallel models, the execution times are proportional to the number of evaluations executed per processor. Therefore, lowering the number of evaluations done by each processor in the parallel executions with respect to the sequential one decreases the time needed to obtain solutions of a given quality. In this regard, the benefit provided by the parallel executions is notable, since the time needed to reach a good quality



(a) NSGA-II



(b) SPEA2

Figure 6.7: Hypervolume for homogeneous models with SLFHL

solution close to the most optimal solution is drastically reduced.

Now, it would be interesting to analyse the heterogeneous island-based models, where each island executes a different MOEA using the Single-Level Filling Heuristic in Length (SLFHL). In theory, the behaviour of a heterogeneous model with two islands, where one island uses NSGA-II and the other SPEA2. However, there are several migration parameters which must be tuned to yield the expected behaviour. Two of the most important aspects are the selection of individuals to migrate and the selection of individuals to replace in each island. In the first attempt, we used a random selection of individuals to migrate and we selected individuals from the worst

HYPERVOL.	Sequential		Parallel 2		Parallel 4		Parallel 8	
	Eval.	Time	Eval.	Time	Eval.	Time	Eval.	Time
80%	333	7.598	150	1.811	130	0.796	65	0.207
85%	493	11.249	240	2.897	210	1.286	155	0.494
90%	800	18.255	516	6.229	388	2.376	291	0.928
95%	2880	65.718	1170	14.125	836	5.121	789	2.516
99%	7125	162.584	5051	60.979	3071	18.811	2175	6.937

Table 6.12: Number of evaluations and time by percentage of hypervolume using homogeneous models

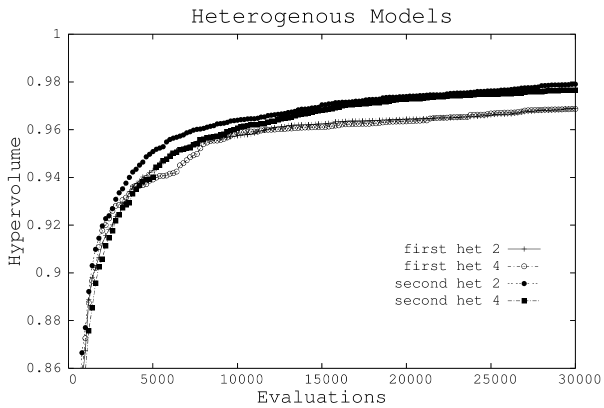


Figure 6.8: Comparison of hypervolume for heterogeneous models

ranks to replace; however, these options did not yield the expected results, so we carried out some tests to obtain better migrations. We eventually obtained better results by selecting random non-dominated individuals to migrate and searching for the nearest individuals in the objective space to replace. Figure 6.8 shows the different behaviours of the heterogeneous models when we used the first kind of selected migrations (labelled as first) and the second kind of selected migrations.

In summary, we verified that every parallelisation of the evolutionary algorithms yielded quality solutions with fewer evaluations, and thus in less time. Some parallelisations are even capable of improving the quality, as in the case of the two-island execution. However, the parallel eight-island model did not manage to reach the final quality obtained by the sequential approach because by increasing the number of islands, the number of evaluations is allocated among more processors, meaning each island makes less progress in each exploration. In other words, each island has less time to explore, which can be verified by seeing the time required by the eight-island parallel execution to reach 99% of the hypervolume of the sequential execution. This time is 23 times less than that used by the sequential execution. We did not pursue this line of enquiry, however, since the multi-level filling heuristics yield very good quality solutions in little time. We also concluded that both the sequential NSGA-II and SPEA2 algorithms improve on the only published results for this multi-objective problem, and that their pararellisations manage to lower the time required, and even to improve the quality of the solutions.

6.5 Validation of the Heuristics and Input Parameters

The results obtained in Section 6.3.2 have shown the competitiveness and good behaviour of the filling heuristics when the 3DPP is formulated as a multi-objective problem, exceeding the results published in [70], where a multi-objective problem is transformed into a mono-objective formulation. However, we believe that a single instance is not enough to develop a computational study. In the literature, there are no compatible instances for our multi-objective formulation, as noted in Section 4.2. For this reason, we developed the Three-Dimensional Multi-Objective Generator (3D-MG) software to generate the multi-objective 3DPP problem instances presented in Section 4.3. Based on the 3D-MG and relying on the Three-Dimensional Multi-Objective Generator in Length (3D-MGL) cutting heuristic, we generated two sets of benchmarks called *1C* and *1A*, presented previously in Section 4.3. Each benchmark consists of a total of 36 different problem instances, which allows us to conduct a more robust and complete computational study.

We concluded in Section 6.3.2 that evolutionary algorithms yield good results for this type of cutting and packing problem. And, moreover, that NSGA-II gives competitive solutions for all of the filling heuristics implemented in Section 3.4.3. The experiments conducted in this section were tested using a single MOEA, the NSGA-II.

Since all of the filling heuristics are executed using the same MOEA, it is interesting to study the behaviour of the four filling heuristics, SLFHL, MLFHL, SLFHD, and MLFHD, when applied to each of the different instances. The goal is to validate that as in the case of the instance proposed by Dereli et al. [70], the best results are those output by the heuristics that consider more than one open level during the packing process (MLFHL, MLFHD).

The 3D-MG was developed to create a range of instances, from identical or ho-

Input parameters	**Definition**
$L \times W \times H$	Dimensions of the container
D	Total number of piece types
D_s	Percent of piece types of D for Sol_1
D_l	Lowest dimensions for a box
D_h	Highest dimensions for a box
S_L	Stowage losses
W_G	Total weight gain of Sol_2
$W_p[]$	Percentage to calculate the weight of the box types of Sol_2

Table 6.13: Properties of the 3D-MG

mogeneous to strongly heterogeneous. However, in this study we considered a set of 72 instances classified from weakly heterogeneous to strongly heterogeneous. The 3D-MG is supplied with a set of input parameters that influence the generation of the various instance types. Table 6.13 shows the set of these input parameters. So as to validate how these parameters influence the complexity of instances, we conducted a set of studies. In these experiments, we study all of the input parameters used to generate the *1C* and *1A* instances.

6.5.1 Description of the Experiments

All of the experimental evaluations were run on a dedicated Debian GNU/Linux operating system with 48 cores, each consisting of one *AMD Opteron 6164HE*. The METCO framework and the problem approach were implemented using C++ and compiled with *gcc 4.7.2*. For the studies we used the NSGA-II. So as to check what the best configuration was for the NSGA-II algorithm, we tested it with different population sizes (20, 25, 50, and 100), mutation rates $(0.1, 0.2,$ and $0.3)$ and crossover rates $(0.7, 0.8,$ and $0.9)$. Table 6.14 shows the configurations used for the NSGA-II. The NSGA-II was used with a population size of 20, a mutation probability of 0.3 and a crossover probability of 0.9. The algorithm was executed with a stopping criterion of 15000 evaluations. From this number of evaluations, the obtained results not improve. Since we are dealing with stochastic algorithms, each execution was repeated 30 times.

The instance generator (3D-MG) was tested by using different values for some of the input parameters that define it. Parameters like S_L and $W_p[]$ were kept constant, as specified in Section 4.4. The remaining input parameters, however, were checked

MOEAs parameters	Tuning values
Population sizes	**20**, 25, 50 and 100.
Mutation rates	0.1, 0, 2 and **0.3**
Crossover rates	0.7, 0.8 and **0.9**
Evaluations	**1500** 30000

Table 6.14: NSGA-II - Initial configuration

3D-MG Input Parameters	Tuning values
$L \times W \times H$	$587 \times 233 \times 220$ (1C) and $1200 \times 233 \times 220$ (1A)
D	$2, 5, 10$, and 15
D_s	60%
$D_l - D_h$	$[5 - 10]\%, [15 - 20]\%$, and $[25 - 30]\%$
S_L	12% (1C) and 8.9% (1A)
W_G	$10\%, 20\%$, and 30%
$W_p[]$	$15.0\%, 25.0\%, 45.0\%, 55.0\%$

Table 6.15: Values of the 3D-MGL parameters

with different values, resulting in benchmarks *1C* and *1A*. Table 6.15 shows the set of values that each of the input parameters can have. The combination of these input parameters gives rise to the 72 instances that will be evaluated in the experiments below (See Appendix C).

6.5.2 Analysis of the Experiments for 1C Problem Instances

In this study, we conduct a computational evaluation using the *1C* problem instances whose definition is given in Section 4.4. The instances evaluated in this section were generated based on the values given in Table 6.15 for each input parameter. For the case of *1C* instances, the value of parameter S_L (Stowage losses) was set to 12%, and the dimensions of the container ($L \times W \times H$) were taken from *UNE 49750 Standard*, and set to $587 \times 233 \times 220$.

6.5.2.1 Evaluation of the Filling Heuristics

In order to compare the behaviour and quality of the results obtained when applying each of the different filling heuristics (SLFHL, MLFHL, SLFHL, and MLFHL) to the various *1C* instances, we conducted an initial experiment. We did a study with each

Instance	Different box types	$[D_l - D_h]$	W_G	S_L	V_{max} (cm^3)	P_{max} $(Kg.)$
$1C$	$2, 5, 10, 15$	$[5-10], [15-20], [25-30]$	10	12%	$3.01e^{07}$	$3.31e^{07}$
$1C$	$2, 5, 10, 15$	$[5-10], [15-20], [25-30]$	20	12%	$3.01e^{07}$	$3.61e^{07}$
$1C$	$2, 5, 10, 15$	$[5-10], [15-20], [25-30]$	30	12%	$3.01e^{07}$	$3.91e^{07}$

Table 6.16: Maximum volume and weight: *1C*

of the 30 executions generated for each filling heuristic and instance combination, and we analysed the final front for each execution in order to select the average-highest volume (Sol.Volume) and the average-highest weight (Sol.Weight). The values for the maximum volume and weight reached by each filling heuristic using *1C* instances are shown in Table 6.16. Note that the volume and weight vary primarily based on the input parameters W_G and S_L. Table 6.17 shows the solution with the average-highest volume and the average-highest weight for each instance when the different filling heuristics are applied. Each of the values shown in Table 6.17 is obtained by calculating the overall average of the best volumes obtained in the final fronts for each of the 30 repetitions (executions). The same process was used to determine the highest average for the weight objective. The volume values shown in Table 6.17 are expressed as a percent of the container volume used, while the weight values, also expressed as a percent, are calculated based on the total weight that can be held inside the container.

As Table 6.17 shows, when MLFHL is used for each of the *1C* instances, it achieves competitive values for both objectives. Thus, if we consider the average-highest volume obtained using each filling heuristic with each instance, we will note how when the SLFHL heuristic is used, better results are obtained when SLFHD is used. This is the case for most of the instances evaluated. The same thing happens when the multiple-level filling heuristics are used, with many of the instances studied, yielding better results in terms of volume when MLFHL is used versus MLFHD. In general, and evaluating each instance independently, Table 6.17 shows how for each instance, the best average-highest volume values were obtained by using MLFHL.

The same study was conducted to consider the average-highest weight (Sol.Weight). In contrast to what happened when evaluating the volume, in this case the average-highest weight values when SLFHD was used were better than those output by the SLFHL heuristic. When studying the average values obtained when using the heuristics with more than one level, we find that MLFHL and MLFHD yield very similar results for most of the instances considered, with the MLFHD approach yielding better results on occasion.

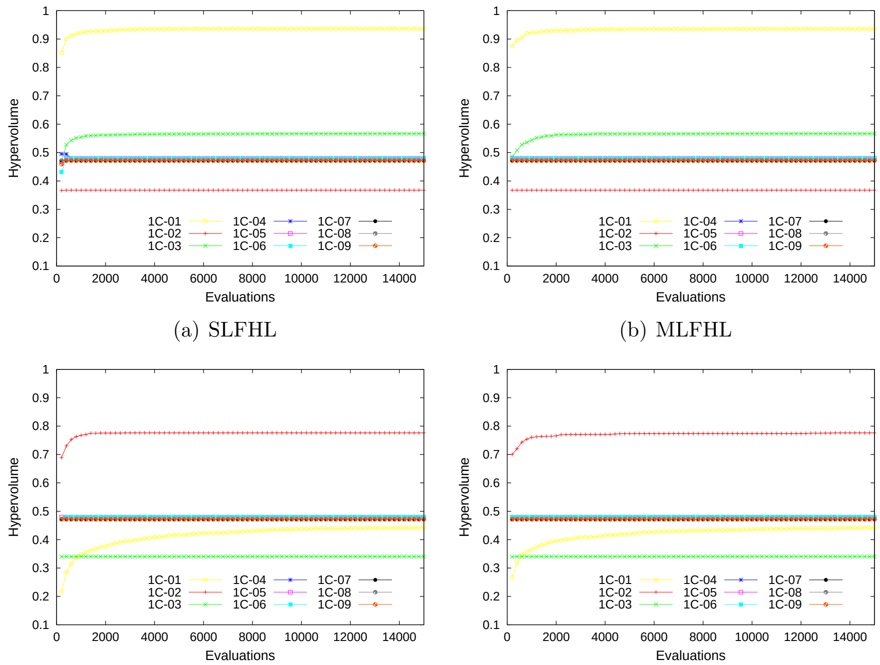
Instance	SLFHL		MLFHL		SLFHD		MLFHD	
	Avg. Volume (%)	Avg. Weight (%)	Avg. Volume (%)	Avg. Weight (%)	Avg. Volume (%)	Avg. Weight (%)	Avg. Volume (%)	Avg. Weight (%)
$1C-01$	**99.61%**	**93.82%**	**99.61%**	**93.82%**	99.04%	51.68%	99.03%	51.71%
$1C-02$	97.53%	83.33%	97.54%	83.33%	**99.16%**	**83.83%**	**99.16%**	**83.83%**
$1C-03$	**99.50%**	**76.48%**	**99.50%**	**76.48%**	98.47%	76.92%	98.47%	76.92%
$1C-04$	98.38%	90.91%	98.38%	90.91%	98.37%	90.91%	98.37%	90.91%
$1C-05$	98.38%	83.33%	98.38%	83.33%	98.37%	83.33%	98.37%	83.33%
$1C-06$	98.38%	76.92%	98.38%	79.92%	98.37%	76.92%	98.37%	76.92%
$1C-07$	98.34%	90.91%	98.35%	90.91%	98.35%	90.91%	98.35%	90.91%
$1C-08$	98.35%	83.33%	98.35%	83.33%	98.35%	83.33%	98.35%	83.33%
$1C-09$	98.35%	76.92%	98.35%	76.92%	98.35%	76.92%	98.35%	76.92%
$1C-10$	96.55%	98.50%	96.93%	**98.88%**	97.01%	97.82%	**97.19%**	97.42%
$1C-11$	96.99%	93.49%	**97.15%**	**93.54%**	95.82%	90.47%	95.88%	90.77%
$1C-12$	97.20%	89.37%	97.37%	**90.17**%	97.42%	87.88%	**97.65%**	85.36%
$1C-13$	**97.56%**	90.91%	**97.56%**	90.91%	91.92%	**97.00%**	92.33%	95.92%
$1C-14$	97.14%	83.46%	**97.19%**	83.62%	94.72%	**92.80%**	95.12%	92.67%
$1C-15$	96.63%	79.54%	**96.68%**	79.79%	93.46%	**94.11%**	93.14%	92.17%
$1C-16$	97.07%	88.94%	**97.89%**	**90.91%**	89.29%	90.04%	89.78%	91.20%
$1C-17$	**97.54%**	83.64%	**97.55%**	83.33%	95.24%	**94.89%**	95.11%	94.01%
$1C-18$	96.87%	78.68%	**97.89%**	76.92%	90.44%	88.67%	90.17%	**89.30%**
$1C-19$	97.69%	95.56%	**97.89%**	**96.31%**	96.91%	95.69%	96.90%	95.04%
$1C-20$	96.92%	96.07%	**97.11%**	95.79%	96.26%	**97.14%**	96.51%	97.00%
$1C-21$	97.27%	91.39%	**97.61%**	90.65%	96.40%	96.38%	96.67%	**96.85%**
$1C-22$	94.32%	92.65%	95.98%	93.94%	96.09%	96.51%	**96.42%**	**96.70%**
$1C-23$	96.52%	88.26%	**97.08%**	89.29%	94.75%	**92.87%**	95.21%	92.65%
$1C-24$	96.55%	89.86%	**96.79%**	91.30%	94.24%	**92.98%**	94.93%	92.58%
$1C-25$	89.23%	86.11%	**94.90%**	**91.51%**	91.08%	86.54%	91.23%	86.24%
$1C-26$	88.25%	81.13%	**94.55%**	**88.30%**	90.72%	85.83%	90.79%	84.06%
$1C-27$	87.42%	76.52%	**95.61%**	**95.61%**	89.24%	87.25%	89.68%	87.11%
$1C-28$	96.04%	**99.94%**	**96.97%**	**99.94%**	95.96%	99.91%	96.28%	**99.94%**
$1C-29$	96.13%	96.89%	**96.72%**	97.60%	94.59%	**97.95%**	94.88%	98.46%
$1C-30$	97.22%	96.61%	**97.71%**	93.29%	95.49%	96.75%	96.24%	**97.33%**
$1C-31$	89.99%	97.09%	**91.70%**	98.10%	90.58%	**98.41%**	91.66%	97.74%
$1C-32$	90.62%	96.53%	91.68%	97.00%	**92.29%**	97.91%	92.13%	**98.69%**
$1C-33$	91.47%	95.82%	92.45%	94.39%	93.23%	97.40%	**93.44%**	**98.50%**
$1C-34$	85.83%	96.51%	88.88%	**97.33%**	87.70%	95.10%	**89.21%**	95.90%
$1C-35$	86.02%	94.27%	87.31%	**95.83%**	87.06%	94.58%	**88.36%**	94.95%
$1C-36$	86.84%	**94.76%**	89.22%	96.04%	87.72%	95.25%	**89.68%**	92.03%

Table 6.17: Comparison of objectives: *1C* problem instances

Given the number of possible combinations of input parameters offered by 3D-MG, and in order to evaluate the behaviour of the filling heuristics in this first study, we used the input parameter D (number of different box types) to discern between the different instance types. Separating the instances in this fashion allowed us to simplify the evaluation and define an order for conducting the experiment. Thus, the input parameter D can take on values of 2, 5, 10, and 15 for the set of instances studied in this section. The first set of instances that will be studied to evaluate the behaviour of the different filling heuristics corresponds to the *1C* instances with $D = 2$. Table 6.18 shows the configurations employed for this set of instances.

Instance	Different box types	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} (Kg.)
$1C-01$	2	$[5-10]$	10	13718	$3.01e^{07}$	$3.31e^{07}$
$1C-02$	2	$[5-10]$	20	2000	$3.01e^{07}$	$3.61e^{07}$
$1C-03$	2	$[5-10]$	30	2662	$3.01e^{07}$	$3.91e^{07}$
$1C-04$	2	$[15-20]$	10	250	$3.01e^{07}$	$3.31e^{07}$
$1C-05$	2	$[15-20]$	20	250	$3.01e^{07}$	$3.61e^{07}$
$1C-06$	2	$[15-20]$	30	250	$3.01e^{07}$	$3.91e^{07}$
$1C-07$	2	$[25-30]$	10	54	$3.01e^{07}$	$3.31e^{07}$
$1C-08$	2	$[25-30]$	20	54	$3.01e^{07}$	$3.61e^{07}$
$1C-09$	2	$[25-30]$	30	54	$3.01e^{07}$	$3.91e^{07}$

Table 6.18: *1C* problem instances with $D = 2$



(a) SLFHL (b) MLFHL

(c) SLFHD (d) MLFHD

Figure 6.9: Hypervolume for *1C* instances with $D = 2$

We used the mean of the hypervolume for 30 independent executions. Figure 6.9 shows the hypervolume for the different heuristics when using the instances shown in Table 6.18 ($D = 2$). The results were highly variable for the set of instances analysed. It is interesting to note when the filling heuristics in length is executed using $1C - 01$, yields very good results, but when the filling heuristics in depth are used, the opposite happens, becoming one of the worst results. The same thing happens when the filling heuristics in length are executed with instance $1C - 03$, the filling heuristics in length give better results. Both instances correspond to ones in which $[D_l - D_h] = [5 - 10]\%$. This behaviour could be related to the probability that exists of leaving more gaps free when using filling heuristics in depth, since the gaps are larger when small boxes are used (Figure 6.5). All configurations must be studied in detail, since there are many parameters to study that could be influencing the results. The instances may also be difficult to solve using the evaluation functions generated in this work. Trying to determine which filling heuristic generally provides the best results for the majority of instances studied leads us to Figure 6.9, which shows how the best quality results are, in general, obtained when using SLFHL and MLFHL. There is no significant difference in the behaviours of SLFHL and MLFHL when applied to the various instances.

The next evaluation was done on instances in which the input parameter D was set equal to 5. Table 6.19 shows the various configurations applied to these instances.

Figure 6.10 shows the hypervolume obtained for each type of filling heuristic, SLFHL, MLFHL, SLFHD, and MLFHD using the different *1C* instances. As we can see, when executing SLFHL and MLFHL with some of the instances, SLFHL and MLFHL yielded acceptable values, but when executed SLFHD and MLFHD with the same instances, the results worsened. There is no significant difference in the behaviours of SLFHL and MLFHL when applied to the various instances. This is most notable when the filling heuristics are executed using instances such as $1C - 11$, $1C - 13$ and $1C - 16$. Each was generated using different parameters, and a priori there is no relationship between them that can be used to determine if they are complex or simple instances. It could simply be that these instances are highly dependent on the parameter values, which influence the operation of the different filling heuristics. In other words, the behaviour varies depending on how the different levels inside the container are opened for the filling heuristics (in length or in depth). Evaluating the behaviour of each different filling heuristics when executed using each of the instances is complex due to the existence of multiple parameters that can influence their performance, such as the effect of the heuristic used.

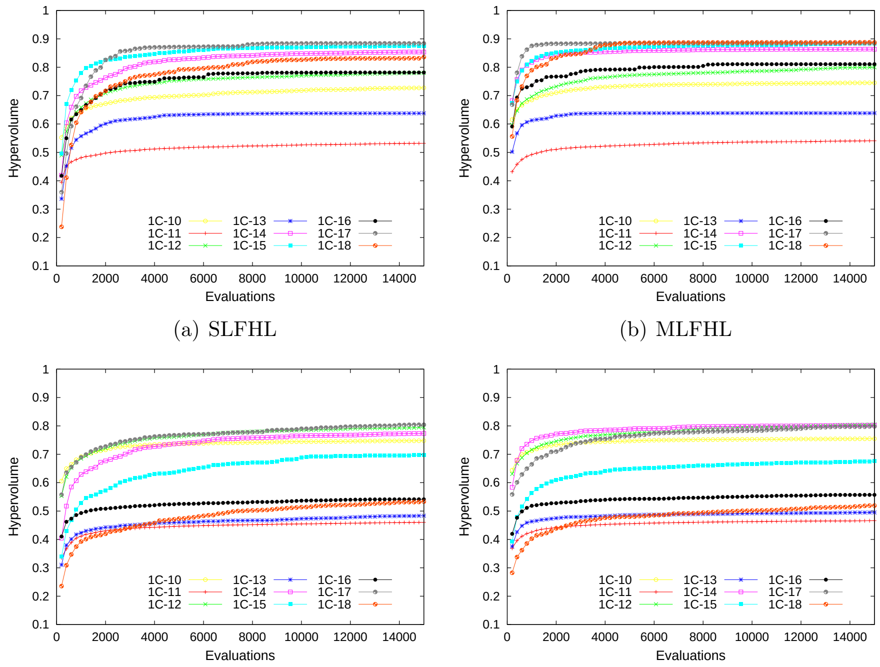
As a result, we evaluated the instances as a whole for the purpose of determining which filling heuristic yields the best quality solutions for the largest set of instances. Figure 6.10 shows that the heuristics that yield the best results are SLFHL and MLFHL, both of which generate levels in length. However, although the results obtained by SLFHD and MLFHD are good, in many cases they cannot match those

Instance	Different box types	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} $(Kg.)$
$1C-10$	5	$[5-10]$	10	3124	$3.01e^{07}$	$3.31e^{07}$
$1C-11$	5	$[5-10]$	20	6894	$3.01e^{07}$	$3.61e^{07}$
$1C-12$	5	$[5-10]$	30	3359	$3.01e^{07}$	$3.91e^{07}$
$1C-13$	5	$[15-20]$	10	380	$3.01e^{07}$	$3.31e^{07}$
$1C-14$	5	$[15-20]$	20	848	$3.01e^{07}$	$3.61e^{07}$
$1C-15$	5	$[15-20]$	30	593	$3.01e^{07}$	$3.91e^{07}$
$1C-16$	5	$[25-30]$	10	180	$3.01e^{07}$	$3.31e^{07}$
$1C-17$	5	$[25-30]$	20	180	$3.01e^{07}$	$3.61e^{07}$
$1C-18$	5	$[25-30]$	30	180	$3.01e^{07}$	$3.91e^{07}$

Table 6.19: *1C* problem instances with $D = 5$

output by the SLFHL and MLFHL heuristics. This behaviour could be associated with two factors. One is the probability that exists of leaving more unoccupied volume when using filling heuristics in depth. This is because as Figure 6.5 shows, the volume that might be left unoccupied is larger than when using heuristics in length. There could also be some relationship between the way that cuts or levels are generated by the different cutting and filling heuristics. Recall that the *1C* instances were generated using 3D-MGL, and the best solutions are associated with SLFHL and MLFHL when are executed with *1C* instances, which generate levels in the same way as the cuts in the 3D-MGL heuristic, with levels or cuts parallel to the x-axis. Figure 6.10 shows good quality results for both the SLFHL and the MLFHL approach when the *1C* instances are evaluated with $D = 5$. After concluding that these are the two filling heuristics that exhibit the best behaviour, it would be interesting to determine if there are any differences between them Thus, with the objective of seeing if there are any differences that could be used to detect which yields better results, the following statistical analysis was conducted.

First we apply the Kolmogorov-Smirnov test in order to check whether the values of the results follow a normal (Gaussian) distribution or not. If so, the Levene test checks for the homogeneity of the variances. If samples have equal variance, an ANOVA test is done; otherwise, a Welch test is performed. For non-Gaussian distributions, the non-parametric Kruskal-Wallis test is used to compare the medians of the algorithms. A confidence level of 95% is considered, which means that the differences are unlikely to have occurred by chance with a probability of 95%. The analysis is performed using the hypervolume metric [231]. Table 6.20 shows the independent statistical comparison between the SLFHL heuristic and the MLFHL heuristics for each *1C* instance with $D = 5$. In other words, each of the different filling heuristics is compared statistically when applied to the same instance. Conducting a statistical study that considers comparisons between different instances



(a) SLFHL (b) MLFHL

(c) SLFHD (d) MLFHD

Figure 6.10: Hypervolume for 1C instances with $D = 5$

Heur.	MLFHL								
	1C-10	1C-11	1C-12	1C-13	1C-14	1C-15	1C-16	1C-17	1C-18
SLFHL	$\downarrow$	$\leftrightarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$

Table 6.20: Statistical comparison between SLFHL and MLFHL with $D = 5$

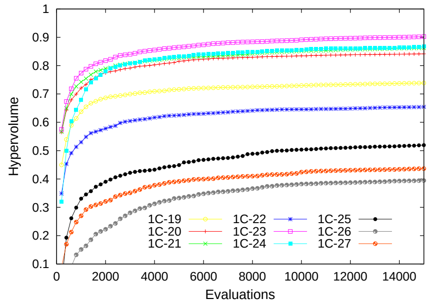
makes no sense, since they were generated differently and other factors besides D would have to be taken into account. The symbol $\uparrow$ is used to denote that differences between the models are statistically significant and that the model in the left column yields a higher median and mean value. In the cases in which the opposite occurs, the symbol $\downarrow$ is used. Finally, for the cases in which the differences were not statistically significant, the symbol $\leftrightarrow$ is used. This study reveals the high capacity of the MLFHL heuristic to achieve good results. Thus, statistically we may conclude that MLFHL behaves better than SLFHL when instances are evaluated with $D = 5$.

Instance	Different box types	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} $(Kg.)$
$1C-19$	10	$[5-10]$	10	7599	$3.01e^{07}$	$3.31e^{07}$
$1C-20$	10	$[5-10]$	20	5946	$3.01e^{07}$	$3.61e^{07}$
$1C-21$	10	$[5-10]$	30	6219	$3.01e^{07}$	$3.91e^{07}$
$1C-22$	10	$[15-20]$	10	924	$3.01e^{07}$	$3.31e^{07}$
$1C-23$	10	$[15-20]$	20	2595	$3.01e^{07}$	$3.61e^{07}$
$1C-24$	10	$[15-20]$	30	1316	$3.01e^{07}$	$3.91e^{07}$
$1C-25$	10	$[25-30]$	10	216	$3.01e^{07}$	$3.31e^{07}$
$1C-26$	10	$[25-30]$	20	216	$3.01e^{07}$	$3.61e^{07}$
$1C-27$	10	$[25-30]$	30	216	$3.01e^{07}$	$3.91e^{07}$

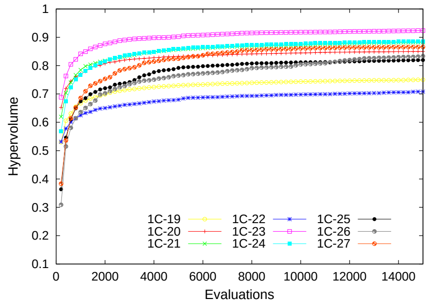
Table 6.21: *1C* problem instances with $D = 10$

So as to continue evaluating the remaining instances in order to determine which filling heuristic (SLFHL, MLFHL, SLFHD, and MLFHD) exhibits the best overall behaviour, without considering the various input parameters involved in creating the different instances, we will now evaluate the behaviour of the filling heuristics when applied to the *1C* instances generated when $D = 10$. Table 6.21 shows the set of instances and the parameters used to generate them.

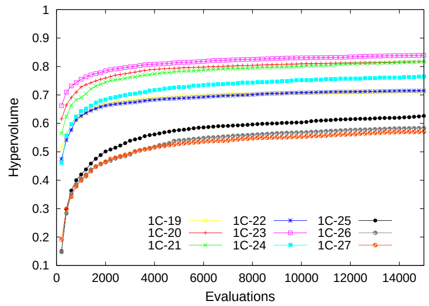
Figure 6.11 shows the hypervolume obtained by each filling heuristic when executed using the different instances with $D = 10$. The configuration used to generate each of the instances is shown in Table 6.21. Figure 6.11 shows how SLFHL, SLFHD and MLFHD behave very similarly when executed using different instances, which improves when using MLFHL. Such is the case, for example, with the different filling heuristics are executed using instances $1C-25$, $1C-26$ and $1C-27$. Other results, however, like those obtained when using $1C-19$ instance, seem unaffected by the type of filling heuristic employed. It is also interesting to note how the obtained results with SLFHL and MLFHL improves when using $1C-23$ Evaluating each instance in this way is complex since there are many parameters that must be independently evaluated for each. For instances with $D = 10$, for example, Figure 6.11 shows that SLFHL and MLFHL yield the best results. In addition, note that the hypervolume results obtained by MLFHL heuristic with the different instances are of better quality than with the other filling heuristics. The results provided by SLFHD and MLFHD are of lower quality than those obtained by SLFHL and MLFHL. Although Figure 6.11 shows the good behaviour of MLFHL, it would be interesting to conduct a statistical analysis that lets us validate this finding. To this end, we carried out a statistical test that compares the two best filling heuristics identified for these instances. Table 6.22 shows a statistical comparison for the different filling heuristics (SLFHL, MLFHL) when applied to the same instance. The symbol $\uparrow$ is used to denote that differences between the models are statistically



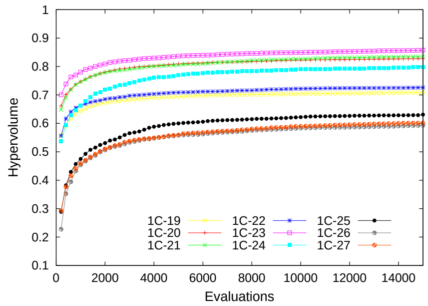
(a) SLFHL



(b) MLFHL



(c) SLFHD



(d) MLFHD

Figure 6.11: Hypervolume for *1C* instances with $D = 10$

Heur.	MLFHL								
	1C-19	1C-20	1C-21	1C-22	1C-23	1C-24	1C-25	1C-26	1C-27
SLFHL	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$

Table 6.22: Statistical comparison between SLFHL and MLFHL with $D = 10$

significant and that the model in the left column yields a higher median and mean value. In those cases in which the opposite occurs, the symbol $\downarrow$ is used. In every case, the good behaviour of MLFHL is confirmed. We may thus conclude that the filling heuristic that yields the best results is MLFHL.

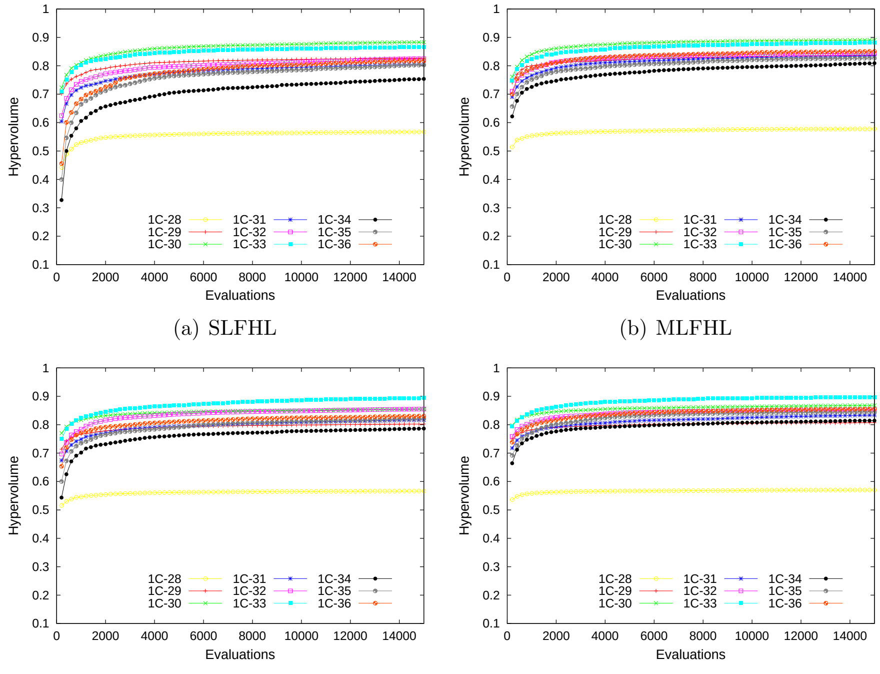
Finally, we evaluate the filling heuristics with instances in which the parameter $D = 15$. Table 6.23 shows the set of instances and the parameters used in 3D-MG to generate them.

Figure 6.12 shows the hypervolume results obtained by the filling heuristics with the set of instances with $D = 15$. In this case, all four filling heuristics (SLFHL,

Instance	Different box types	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} $(Kg.)$
$1C-28$	15	$[5-10]$	10	9656	$3.01e^{07}$	$3.31e^{07}$
$1C-29$	15	$[5-10]$	20	6156	$3.01e^{07}$	$3.61e^{07}$
$1C-30$	15	$[5-10]$	30	9775	$3.01e^{07}$	$3.91e^{07}$
$1C-31$	15	$[15-20]$	10	1108	$3.01e^{07}$	$3.31e^{07}$
$1C-32$	15	$[15-20]$	20	1185	$3.01e^{07}$	$3.61e^{07}$
$1C-33$	15	$[15-20]$	30	1047	$3.01e^{07}$	$3.91e^{07}$
$1C-34$	15	$[25-30]$	10	405	$3.01e^{07}$	$3.31e^{07}$
$1C-35$	15	$[25-30]$	20	405	$3.01e^{07}$	$3.61e^{07}$
$1C-36$	15	$[25-30]$	30	405	$3.01e^{07}$	$3.91e^{07}$

Table 6.23: *1C* problem instances with $D = 15$

MLFHL, SLFHD and MLFHD) yield good quality results. Most of the results cluster around 0.9, which assures that the results are very close to the optimum. For one of the instances, however, it seems that filling heuristics have problems obtaining good results, possibly due to its complexity. The instance is $1C - 28$, which considers small boxes. Also, the degree of contradiction between the objectives is low, given that $W_G = 10$. Since the input parameter W_G is crucial to determining the total weight that can be loaded into the container, the smaller this parameter, the smaller the difference between the container's volume and its weight. But as with the other studies carried out earlier, it would be difficult to determine the real cause for the poor results exhibited by this instance. Due to the similarity in the results observed for every type of filling heuristic, it would be interesting to carry out a statistical analysis that can let us ascertain if there is some statistical difference between them. Based on the good behaviour exhibited by MLFHL in the studies with $D = 2, 5$ and 10, we propose a statistical test in which we compare SLFHL, SLFHD and MLFHD with MLFHL, considering each instance independently. Table 6.24 shows said comparison. The symbol $\uparrow$ is used to denote that differences between the models are statistically significant and that the model in the left column yields a higher median and mean value. In the cases in which the opposite occurs, the symbol $\downarrow$ is used. For the set of instances analysed, we can see how statistically, MLFHL exhibits better behaviour than the SLFHL heuristic and the SLFHD heuristic. However, when we compare the two filling heuristics assuming more than one open level during the container packing phase, we see how, for certain instances, MLFHD yields results that are not statistically significant. We thus cannot conclude that its behaviour is worse than when MLFHL is used. Its behaviour is even better when MLFHL is executed with instances $1C - 33$ and $1C - 35$. However, in general we see that MLFHL exhibits very good behaviour, statistically speaking, versus the remaining filling heuristics.



(a) SLFHL

(b) MLFHL

(c) SLFHD

(d) MLFHD

Figure 6.12: Hypervolume for *1C* instances with $D = 15$

Heur.	MLFHL								
	1C-28	1C-29	1C-30	1C-31	1C-32	1C-33	1C-34	1C-35	1C-36
SLFHL	↓	↓	↓	↓	↓	↓	↓	↓	↓
SLFHD	↓	↓	↓	↓	↔	↑	↓	↔	↓
MLFHD	↓	↓	↓	↔	↔	↑	↔	↑	↔

Table 6.24: Statistical comparison between SLFHL, SLFHD, MLFHD and MLFHL with $D = 15$

Of the studies conducted on the *1C* problem instances, we can conclude that the filling heuristic with the best overall results is the MLFHL heuristic. However, we cannot ignore the possible relationship that might exist between the cutting heuristics used by 3D-MG and the filling heuristics employed to evaluate the solutions when both use the same procedure to create levels inside the container. Although the studies conducted do not allow us to conclude if such a relationship exists, it

3D-MG - Input parameters	Tuning values
$L \times W \times H$	$587 \times 233 \times 220$
D	$2, 5, 10$, and 15
D_s	60%
$D_l - D_h$	$[5-10]\%, [15-20]\%$, and $[25-30]\%$
S_L	12%
W_G	$10\%, 20\%$, and 30%
$W_p[]$	$15.0\%, 25.0\%, 45.0\%, 55.0\%$

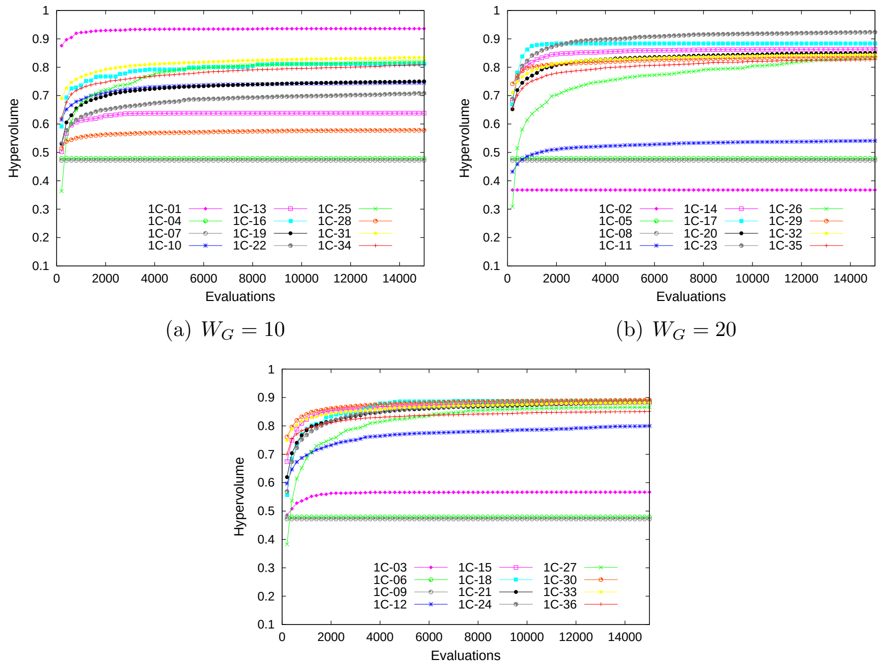
Table 6.25: Parameter values for generating the *1C* instances

is interesting to note that when the cutting heuristic 3D-MGL is used to generate the 1C instances, higher quality results are obtained when it is combined with the SLFHL and MLFHL filling heuristics.

6.5.2.2 Evaluation of the Input Parameters

Based on the above studies, we have determined that, in general, the filling heuristic that yields the best results for the set of instances evaluated (*1C* instances) is the MLFHL heuristic. The remaining experiments will thus be based on the results obtained when the various instances are applied on the MLFHL heuristic using NSGA-II. In this way, the experiments that follow attempt to determine what input parameter configuration is required for 3D-MG to generated problems with different complexities. Table 6.25 shows the set of input parameters whose values were used to generate the *1C* problem instances. As we can see, parameters $L \times W \times H$, S_L and $W_p[]$ are constant. The comparisons were therefore made using the variable input parameters. We also intend to determine if changing these parameters has an effect on the complexity of the instances generated.

The first comparison involves the parameter W_G (total weight gain of Sol_2). W_G represents the weight increase that results from the optimal weight solution (Sol_2) and corresponds to the container's maximum (tare) weight, P_{max}. Thus, the greater the value used for this parameter, the higher the degree of contradiction between the two objectives evaluated in the 3DPP studied in this thesis, namely, volume and weight. This is because more weight can be packed into the container for the same container volume. Thus, the parameter W_G has a direct influence on the degree of contradiction in relation to the container. For the instances used, this input parameter can take on values of $W_G = 10, 20$, and 30. Figure 6.13 shows the hypervolume reached by MLFHL heuristic when applied to the set of problem



(a) $W_G = 10$

(b) $W_G = 20$

(c) $W_G = 30$

Figure 6.13: Hypervolume for different values of W_G

instances with the different settings for the input parameter W_G. To simplify the experiment, we separated the instances by considering the value assigned to W_G. All of the instances generated with $W_G = 10$ are shown together. The same applies to $W_G = 20$ and $W_G = 30$. We can see how when MLFHL is executed using the instances with the input parameter $W_G = 10$, the quality of the solutions is well below those exhibited in those instances where $W_G = 20$ and $W_G = 30$. This behaviour occurs because with higher values of W_G, the container can be loaded with more weight, thus achieving higher quality solutions for the two objectives considered, volume and weight. This is because the search space is larger and the solutions are more dispersed, which allows obtaining higher quality solutions. Thus, the solutions obtained by MLFHL when using instances with $W_G = 20$ and $W_G = 30$ are fairly good in both cases. In Figure 6.13, though no clear difference is apparent between the two values of W_G, we can see how when MLFHL is using with instances with a value of $W_G = 30$, the solutions cluster around 0.9, while if a value of $W_G = 20$

WG	$W_G = 20/W_G = 30$		
$D = 5$	$1C - 11/1C - 12$	$1C - 14/1C - 15$	$1C - 17/1C - 18$
	$\downarrow$	$\downarrow$	$\downarrow$
$D = 10$	$1C - 20/1C - 21$	$1C - 23/1C - 24$	$1C - 26/1C - 27$
	$\downarrow$	$\uparrow$	$\downarrow$
$D = 15$	$1C - 29/1C - 30$	$1C - 32/1C - 33$	$1C - 35/1C - 36$
	$\downarrow$	$\leftrightarrow$	$\downarrow$

Table 6.26: Statistical comparison between $W_G = 20$ and $W_G = 30$

is used, the solutions, despite their good quality, are more scattered. It is not simple to use this method, however, to determine which value for the input parameter W_G generates instances of greater or lesser complexity.

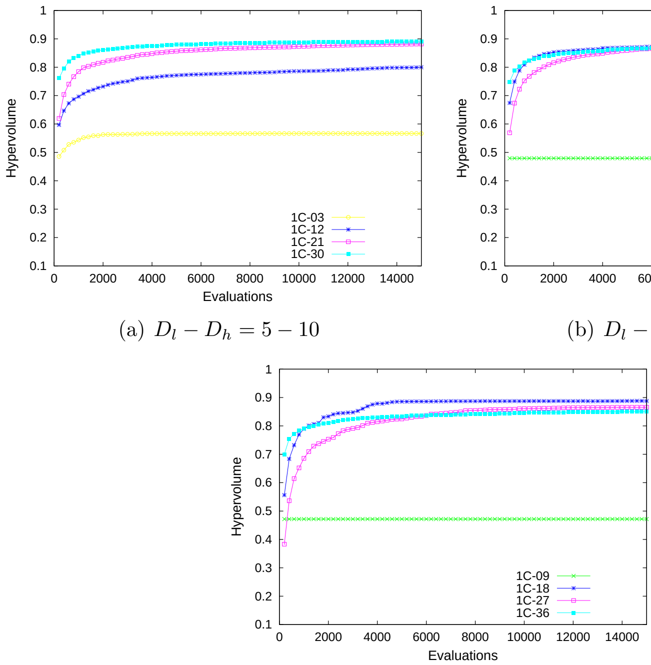
As a result, it would be interesting to conduct a statistical analysis that can help us identify any significant differences that will allow us to determine the value for this input parameter, W_G, which allow to generate instances with different complexities. We did so by showing the robustness of the input parameter for different values. Table 6.26 shows a statistical comparison between the results obtained by MLFHL with the different instances where the only parameter that changes is W_G. Any other comparison would be meaningless, since the instances were generated using very different configurations. For example, the results obtained by MLFHL with the instance $1C-11$, in which $W_G = 20$, is only compared to the results when MLFHL is executed using $1C - 12$, where $W_G = 30$, since the remaining input parameters are the same. The symbol $\uparrow$ is used to denote that differences between the models are statistically significant and that the model more to left ($W_G = x/W_G = Y$) has a higher median and mean value. In the cases in which the opposite occurs, the symbol $\downarrow$ is used. In this study we did not include instances with $D = 2$, since they are not statistically comparable to the rest. Note how MLFHL with instances with $W_G = 30$ behave better statistically than with $W_G = 20$. We thus conclude that MLFHL obtained the best value when the instances are generated with $W_G = 30$. In addition, we can think that problem instances with $W_G = 30$ are easier to solve for our filling heuristics than those using $W_G = 20$. Which seem to have a greater degree of difficulty.

Having analysed the behaviour the MLFHL heuristic for the instances with different values of input parameter W_G, and having determined that MLFHL yields the best quality solutions with the set of instances with $W_G = 30$, we will now evaluate another of the parameters used to generate the *1C* problem instances. The remaining studies will this consider MLFHL as the filling heuristic executed using NSGA-II, and employing only those *1C* instances that were generated with input parameter $W_G = 30$. This experiment thus focuses on analysing the behaviour of the results obtained by MLFHL when it is executed with instances with different values

Instance	Different box types	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} $(Kg.)$
$1C-03$	2	$[5-10]$	30	2662	$3.01e^{07}$	$3.91e^{07}$
$1C-06$	2	$[15-20]$	30	250	$3.01e^{07}$	$3.91e^{07}$
$1C-09$	2	$[25-30]$	30	54	$3.01e^{07}$	$3.91e^{07}$
$1C-12$	5	$[5-10]$	30	3359	$3.01e^{07}$	$3.91e^{07}$
$1C-15$	5	$[15-20]$	30	593	$3.01e^{07}$	$3.91e^{07}$
$1C-18$	5	$[25-30]$	30	180	$3.01e^{07}$	$3.91e^{07}$
$1C-21$	10	$[5-10]$	30	6219	$3.01e^{07}$	$3.91e^{07}$
$1C-24$	10	$[15-20]$	30	1316	$3.01e^{07}$	$3.91e^{07}$
$1C-27$	10	$[25-30]$	30	216	$3.01e^{07}$	$3.91e^{07}$
$1C-30$	15	$[5-10]$	30	9775	$3.01e^{07}$	$3.91e^{07}$
$1C-33$	15	$[15-20]$	30	1047	$3.01e^{07}$	$3.91e^{07}$
$1C-36$	15	$[25-30]$	30	405	$3.01e^{07}$	$3.91e^{07}$

Table 6.27: *1C* problem instances with $W_G = 30$

for the input parameters D_l and D_h. Input parameters D_l and D_h (Table 6.15) determine the lowest and highest dimensions that a box type can have with respect to the dimensions of the container, respectively. The set of instances evaluated in this experiment are shown in Table 6.27. The possible values of $[D_l - D_h]$ are $[5-10]\%$, $[15-20]\%$ and $[25-30]\%$. These parameters are studied as if they were a single parameter ($[D_l - D_h]$). Combining the values of D_l and D_h is not possible for the implementation carried out for 3D-MG. A set of boxes is considered to be small when its dimensions are between $[5-10]\%$ of the length, height and width of the container, and large when its dimensions are between $[25-30]\%$. Boxes outside these ranges are regarded as medium sized. The hypervolume [231] achieved for MLFHL with each instance is shown in Figure 6.14. This one shows the hypervolume obtained by MLFHL with the *1C* instances with different values for the input parameters D_l and D_h. So as to simplify the evaluation of these input parameters, we divided the instances evaluated in this experiment based on D_l and D_h. All of the instances with same value of D_l and D_h were grouped together. When MLFHL is executed with instances where the value $[D_l - D_h] = [15-20]\%$, more promising results are obtained than for the other possible values of the input parameters evaluated in this experiment. This is contrary to what one might think initially, that the best results would be obtained by MLFHL with the combinations $[5-10]\%$ and $[25-30]\%$, in which it would be simpler to fill in the gaps. The study shows, however, that the best results are those obtained by MLFHL with medium-sized boxes. Thus, we can conclude that the instances that use $WG = 30$ and $[D_l - D_h] = [15-20]\%$ combination are easier to solve. Whilst the reamining instances are more complex, and the filling heuristic has more problems to reach the optimal solution. In conclusion,



(a) $D_l - D_h = 5 - 10$ (b) $D_l - D_h = 15 - 20$

(c) $D_l - D_h = 15 - 30$

Figure 6.14: Comparison between different box sizes

	$[5-10]$ 1C-12	$[15-20]$ 1C-15	$[25-30]$ 1C-18
$[5-10]$	$\leftrightarrow$	$\downarrow$	$\downarrow$
$[15-20]$	$\uparrow$	$\leftrightarrow$	$\uparrow$
$[25-30]$	$\uparrow$	$\downarrow$	$\leftrightarrow$

Table 6.28: Statistical comparison for $[D_l - D_h]\%$ with $D = 5$ and $W_G = 30$

MLFHL achieves the highest quality solutions when it is executed with the instances which were created using $WG = 30$ and $[D_l - D_h] = [15 - 20]\%$, the reason being that with this box size, there is a lower probability of leaving unoccupied spaces. This is because if only small boxes are used to pack the container, it could result in leaving many small, scattered spaces unfilled, leading to a greater loss of volume inside the container. Something similar occurs with larger boxes, since small gaps could be created that are not capable of being filled by any of the large boxes.

	$[5-10]$ 1C-21	$[15-20]$ 1C-24	$[25-30]$ 1C-27
$[5-10]$	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$
$[15-20]$	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$
$[25-30]$	$\downarrow$	$\leftrightarrow$	$\leftrightarrow$

Table 6.29: Statistical comparison for $[D_l - D_h]\%$ with $D = 10$ and $W_G = 30$

	$[5-10]$ 1C-30	$[15-20]$ 1C-33	$[25-30]$ 1C-36
$[5-10]$	$\leftrightarrow$	$\downarrow$	$\downarrow$
$[15-20]$	$\uparrow$	$\leftrightarrow$	$\uparrow$
$[25-30]$	$\uparrow$	$\downarrow$	$\leftrightarrow$

Table 6.30: Statistical comparison for $[D_l - D_h]\%$ with $D = 15$ and $W_G = 30$

We thus conclude that MLFHL obtained the best results with medium-sized boxes, since although the likelihood of finding gaps is the same, they will be smaller in both quantity and volume. In order to provide statistical reliable results, suitable statistical analyses must be performed. The statistical analysis for the instances with $D = 2$ are not included because they are not statistically comparable. Table 6.28, Table 6.29 and Table 6.30 show whether the row configuration is statistically better ($\uparrow$), not different ($\leftrightarrow$), or worse ($\downarrow$) than the corresponding column combination. Table 6.28 shows the statistical analysis for MLFHL with $D = 5$ and $W_G = 30$, the goal being to compare those instances where the variable value corresponds to the input parameters $[D_l - D_h]$. Any other comparison is not feasible since the instances were generated using different input parameters. The same statistical comparison is shown in Table 6.29, in this case for $D = 10$ and $W_G = 30$. And finally, Table 6.30 shows the comparison between the results obtained by MLFHL with *1C* instances with $D = 15$ and $W_G = 30$. All of the statistical analyses conducted show how the MLFHL heuristic with the instances with the configuration $D = 5, 10, 15$, $W_G = 30$ and $[D_l - D_h] = [15 - 20]$ yields statistically better results than with the instances that use the combination with $[25 - 30]\%$. We thus conclude that mlfhl yield the highest quality solutions with the instances generated using input parameters $W_G = 30$ and $[D_l - D_h] = [15 - 20]$. As these are simpler instances, or easier to solve by the designed filling heuristic.

Having analysed the behaviour of the input parameters W_G and $[D_l - D_h]$ used by 3D-MG to generate instances, and having concluded that the values for both parameters $W_G = 30$ and $[D_l - D_h] = [15 - 20]$ are the ones that allow to yield the best solutions for the MLFHL heuristic. We will now proceed to analyse the last of the variable input parameters used to create the benchmark set *1C*, namely input parameter D. Input parameter D determines the number of different box types that

Instance	Different box types	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} $(Kg.)$
$1C - 06$	2	$[15 - 20]$	30	250	$3.01e^{07}$	$3.91e^{07}$
$1C - 15$	5	$[15 - 20]$	30	593	$3.01e^{07}$	$3.91e^{07}$
$1C - 24$	10	$[15 - 20]$	30	1316	$3.01e^{07}$	$3.91e^{07}$
$1C - 33$	15	$[15 - 20]$	30	1047	$3.01e^{07}$	$3.91e^{07}$

Table 6.31: *1C* problem instances with $W_G = 30$ and $[D_l - D_h] = [15 - 20]\%$

comprise each instance. The possible values that D can have are shown in Table 6.25, these being $D = 2, 5, 10, 15$. The next experiment will be conducted using only those *1C* instances that were generated using as input parameters $W_G = 30$ and $[D_l - D_h] = [15 - 20]$. Table 6.31 shows the set of instances with the corresponding configurations that will be used to analyse input parameter D.

Figure 6.15 shows the hypervolume when applying the MLFHL heuristic with instances with $W_G = 30$ and $[15 - 20]\%$ and varying input parameter D. As we can see, the instances with $D = 5, D = 10$ and $D = 15$ behave very similarly. Opposite results are obtained by MLFHL when instances have $D = 2$. Once more we may conclude that the instances with $D = 2$ are difficult to evaluate for the filling heuristic proposed. A more in-depth study would be required to determine if their behaviour could be related to the complexity of the instance, or if there is another reason why low quality results are being obtained compared to the other instances compared in this study. For all other instances different from $D = 2$, no clear conclusions can be drawn since they all exhibit very similar behaviour. In order to provide the results with confidence, we carried out a statistical study, omitting the results obtained when using the instance with $D = 2$ since it is not statistically comparable. Table 6.32 shows the statistical comparison for the results obtained using MLFHL with different instances whose value D is the variable input parameter. The remaining variable input parameters were set to the values that allows to obtain the best results for MLFHL, as determined in the previous studies. This analysis shows no significant results, however. We thus conclude that the MLFHL heuristic behaves well independently of this parameter for the values of W_G and $D_l - D_h$ assigned.

As for the input parameters used by 3D-MG, we have determined that when MLFHL is executed with the larger the W_G parameter, the better the quality of the results. In other words, they are easy to solve by our filling heuristic. As concerns input parameters D_l and D_h, we concluded that when MLFHL is executed with medium-sized boxes, $[15 - 20]$, yield higher quality solutions than those obtained with instances that considered small ($[5-10]$) and large ($[25-30]$) boxes. Thus, the combination of $W_G = 30$ and $[D_l - D_h] = [15-20]$ generate simple instances. While

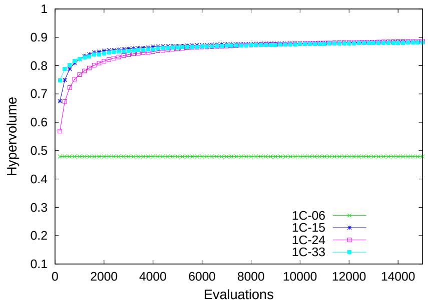


Figure 6.15: Comparison between different values of D with $W_G = 30$ and $[15-20]\%$

$[D_l - D_h]$	$[5-10]$ 1C-15	$[15-20]$ 1C-24	$[25-30]$ 1C-33
1C-15	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$
1C-24	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$
1C-33	$\leftrightarrow$	$\leftrightarrow$	$\leftrightarrow$

Table 6.32: Statistical comparison between D using $W_G = 30$ and $[15-20]\%$

the other combinations generate more complex instances. One possible cause could be that when small and large boxes are involved, the likelihood of creating holes is higher due to the proportion of said boxes. In the end, we were unable to draw any conclusions as to the value for input parameter D. We determined that its value may not be a determining factor for the input parameters considered ($W_G = 30$ and $[D_l - D_h] = [15-20]$), meaning that good solutions of the MLFHL heuristic can be obtained regardless of D.

6.5.3 Analysis of the Experiments for 1A Problem Instances

For the next computational studies, we used the *1A* problem instances defined in Section 4.4. The values used for the different input parameters of Three-Dimensional Multi-Objective Generator (3D-MG) are shown in Table 6.15. For the case of the *1A* instances, the value of parameter S_L (stowage losses) was set to 8.9%, and the dimensions of the container ($L \times W \times H$) were taken from the *UNE 49750 Standard*, and set to $1200 \times 233 \times 220$.

Insstance	Different box types	$[D_l - D_h]$	W_G	V_{max} (cm^3)	P_{max} ($Kg.$)
$1A$	$2,5,10,15$	$[5-10],[15-20],[25-30]$	10	$6.15e^{07}$	$6.77e^{07}$
$1A$	$2,5,10,15$	$[5-10],[15-20],[25-30]$	20	$6.15e^{07}$	$7.38e^{07}$
$1A$	$2,5,10,15$	$[5-10],[15-20],[25-30]$	30	$6.15e^{07}$	$7.90e^{07}$

Table 6.33: Maximum volume and weight: *1A*

6.5.3.1 Evaluation of the Filling Heuristics

So as to compare the quality of the results yielded by each of the different filling heuristics (SLFHL, MLFHL, SLFHL, and MLFHL) for the instances evaluated in this section, we conducted a series of experiments. In the first, we evaluated the final fronts for each of the 30 executions run for each of the instances applied to the various filling heuristics in order to select the average-highest volume (Sol.Volume) and the average-highest weight (Sol.Weight). The maximum volume and weight values that can be obtained for each filling heuristic with the instances are shown in Table 6.33. The maximum weight P_{max} allowed for each instance type depends on the input parameter W_G (total weight gain of Sol_2). Thus we can see how increasing W_G results in an increase in the maximum allowed weight (P_{max}). The average values obtained for volume and weight for the set analysed are shown in Table 6.34. To calculate the average-highest volume, we used the best volume values obtained in each of the final fronts for the 30 executions and calculated the average, expressed as a percent. The same calculation was carried out to obtain the average-highest weight. The volume results from considering the total space used inside the container, while the weight is obtained based on the total weight stored inside the container.

The results obtained by each filling heuristic with each instance has to be evaluated separately. This is because each instance was generated using different parameters, making them computationally incomparable. With this premise in mind, and seeing the results shown in Table 6.34, we can determine that the average-highest volumes are obtained when using the MLFHL filling heuristics for most of the instances. In general, when comparing filling heuristics SLFHL and SLFHD with each instance, the average-highest volume is obtained when the SLFHL heuristic is used. The same thing happens when comparing MLFHL and MLFHD, in which the average-highest volume is generally obtained with MLFHL for each of the various instances. Finally, if we compare the average volumes when using SLFHL and MLFHL on the one hand, and SLFHD and MLFHD on the other, we find that although there is no significant difference between the results, the highest values appear when the filling heuristics MLFHL and MLFHD are used. The same study

Instance	SLFHL		MLFHL		SLFHD		MLFHD	
	Avg. Volume (%)	Avg. Weight (%)	Avg. Volume (%)	Avg. Weight (%)	Avg. Volume (%)	Avg. Weight (%)	Avg. Volume (%)	Avg. Weight (%)
$1A-01$	98.71%	90.91%	98.71%	90.91%	98.71%	90.91%	98.71%	90.91%
$1A-02$	**97.81%**	**83.89%**	**97.82%**	**83.89%**	96.07%	83.33%	96.07%	83.33%
$1A-03$	**93.07%**	60.22%	**93.08%**	60.28%	89.37%	**76.92%**	89.38%	**76.92%**
$1A-04$	96.07%	90.91%	96.07%	90.91%	96.07%	90.91%	96.07%	90.91%
$1A-05$	98.71%	83.33%	98.71%	83.33%	98.71%	83.33%	98.71%	83.33%
$1A-06$	98.71%	76.92%	98.71%	76.92%	98.71%	76.92%	98.71%	76.92%
$1A-07$	98.69%	90.91%	98.69%	90.91%	98.69%	90.91%	98.69%	90.91%
$1A-08$	98.69%	83.33%	98.69%	83.33%	98.69%	83.33%	98.69%	83.33%
$1A-09$	98.69%	76.92%	98.69%	76.92%	98.69%	76.92%	98.69%	76.92%
$1A-10$	**98.60%**	92.70%	98.51%	93.78%	96.21%	93.89%	96.28%	**95.22%**
$1A-11$	98.43%	93.19%	**98.95%**	93.94%	95.59%	96.63%	95.67%	**96.67%**
$1A-12$	97.23%	93.45%	**97.43%**	94.20%	93.35%	98.54%	93.52%	**98.70%**
$1A-13$	97.96%	**90.96%**	**97.98%**	90.91%	96.77%	92.06%	97.19%	90.48%
$1A-14$	98.03%	82.67%	**98.27%**	83.33%	95.05%	**84.96%**	95.07%	84.54%
$1A-15$	**98.60%**	79.14%	**98.59%**	79.57%	93.42%	95.34%	93.44%	**96.62%**
$1A-16$	97.58%	90.28%	97.83%	90.91%	98.53%	**92.10%**	99.04%	**91.89%**
$1A-17$	97.94%	83.33%	97.94%	83.33%	99.20%	99.16%	**99.88%**	**99.56%**
$1A-18$	**98.29%**	76.95%	**98.29%**	76.92%	92.51%	**93.36%**	92.85%	89.51%
$1A-19$	96.78%	**96.73%**	**97.72%**	95.35%	96.15%	95.62%	96.47%	94.93%
$1A-20$	**97.93%**	93.54%	97.89%	92.50%	95.67%	95.25%	95.92%	**97.11%**
$1A-21$	97.05%	85.21%	**97.29%**	89.54%	95.05%	**92.26%**	95.47%	88.83%
$1A-22$	96.39%	92.58%	**97.43%**	94.24%	91.20%	94.66%	91.72%	**97.48%**
$1A-23$	94.53%	86.94%	**95.10%**	85.66%	94.21%	88.20%	94.89%	**89.93%**
$1A-24$	94.53%	91.13%	**95.67%**	90.29%	92.47%	94.36%	93.22%	**95.93%**
$1A-25$	**94.57%**	90.96%	94.61%	93.81%	92.35%	97.17%	93.11%	**97.85%**
$1A-26$	95.43%	84.94%	**96.11%**	83.84%	88.50%	96.26%	89.61%	**97.20%**
$1A-27$	93.96%	88.47%	**95.20%**	85.06%	90.40%	92.13%	92.09%	**96.83%**
$1A-28$	97.78%	98.86%	**98.29%**	99.08%	96.28%	99.50%	96.74%	**99.59%**
$1A-29$	97.18%	93.69%	**97.51%**	93.66%	95.53%	**98.91%**	95.23%	97.81%
$1A-30$	**97.83%**	93.46%	**97.83%**	93.80%	96.09%	96.37%	95.76%	**98.14%**
$1A-31$	90.61%	96.52%	91.86%	93.70%	92.26%	97.70%	**92.70%**	**98.62%**
$1A-32$	91.22%	97.18%	**93.36%**	97.30%	91.39%	98.08%	91.79%	**99.13%**
$1A-33$	90.62%	**95.49%**	92.56%	95.16%	93.06%	94.54%	**93.67%**	93.44%
$1A-34$	85.12%	**96.97%**	**87.67%**	95.07%	81.71%	89.17%	83.22%	91.64%
$1A-35$	84.58%	95.26%	**87.74%**	**98.34%**	79.83%	93.46%	82.21%	93.80%
$1A-36$	86.67%	94.50%	**98.31%**	**96.03%**	83.04%	86.18%	85.24%	86.02%

Table 6.34: Comparison of objectives: *1A* problem instances

was repeated for the average-highest weight (Sol.Weight). In this case we find that for the set of filling heuristics with the instances studied, the average-highest weight values were obtained when using MLFHD. SLFHD also yielded better results on average than SLFHL for most of the *1A* instances. The same thing happens when we compare MLFHL and MLFHD, where the highest values are found, though not always, for MLFHD.

Given the magnitude of the possible combinations of input parameters offered by 3D-MG, and so as to evaluate the behaviour of the filling heuristics for this first study, we relied on input parameter D (number of different box types) to allow us

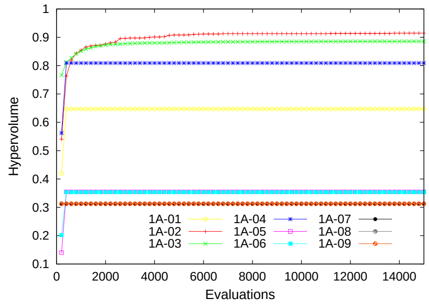
Instance	Different box types	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} (Kg.)
$1A-01$	2	$[5-10]$	10	2000	$6.15e^{07}$	$6.77e^{07}$
$1A-02$	2	$[5-10]$	20	3456	$6.15e^{07}$	$7.38e^{07}$
$1A-03$	2	$[5-10]$	30	4394	$6.15e^{07}$	$7.99e^{07}$
$1A-04$	2	$[15-20]$	10	432	$6.15e^{07}$	$6.77e^{07}$
$1A-05$	2	$[15-20]$	20	250	$6.15e^{07}$	$7.38e^{07}$
$1A-06$	2	$[15-20]$	30	250	$6.15e^{07}$	$7.99e^{07}$
$1A-07$	2	$[25-30]$	10	54	$6.15e^{07}$	$6.77e^{07}$
$1A-08$	2	$[25-30]$	20	54	$6.15e^{07}$	$7.38e^{07}$
$1A-09$	2	$[25-30]$	30	54	$6.15e^{07}$	$7.99e^{07}$

Table 6.35: *1A* problem instances with $D = 2$

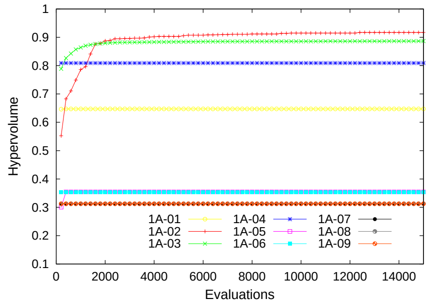
to differentiate between the various instance types. Separating the instances in this fashion allows us to simplify the evaluation. We will thus start by studying the instances generated by letting $D = 2$ for the maximum number of box types, and then continue with the remaining possible values considered for the input parameter: 5, 10, and 15. Table 6.35 shows the configurations used for this set of instances.

Figure 6.16 shows the mean of the hypervolume for 30 independent executions carried out for the different heuristics when using the instances given in Table 6.35 ($D = 2$). The results are difficult to analyse, since as the table shows, the behaviour of all four filling heuristics is very similar for the various instances. Generally speaking, the atypical behaviour of the filling heuristics with these instances could result from the inability of the filling heuristics employed to efficiently solve these instances due to their complex nature. This assumption would require a more in-depth study that considers every possible combination of input parameters, filling heuristics and even the effect of the MOEAs.

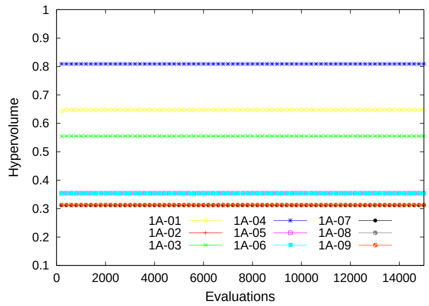
In keeping with the specified order, the next study is conducted considering the set of *1A* instances generated by setting input parameter $D = 5$. Table 6.36 shows the different configurations applied for these instances. Figure 6.17 shows the hypervolume obtained by each type of filling heuristic, SLFHL, MLFHL, SLFHD, and MLFHD with each instances ($D = 5$). In general, SLFHL and MLFHL clearly yield very high quality solutions for all of the instances studied, with the solutions all clustered very close to the optimal. In contrast, when SLFHD and MLFHD are applied, though they obtain good values in many cases, they are below those of the previous heuristics. This is clearly evident when evaluating heuristics when using instances like $1A - 12$, $1A - 15$ and $1A - 18$, where we can see how applying SLFHL and MLFHL yields high quality solutions compared to the poorer results obtained by SLFHD and MLFHD. This behaviour could be linked to two factors. One is the probability of leaving more unoccupied gaps when using filling heuristics



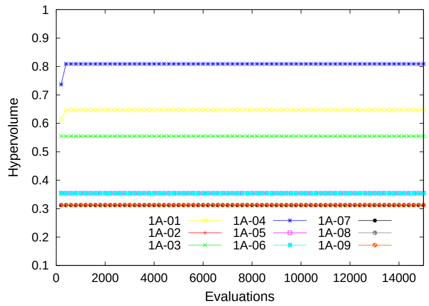
(a) SLFHL



(b) MLFHL



(c) SLFHD

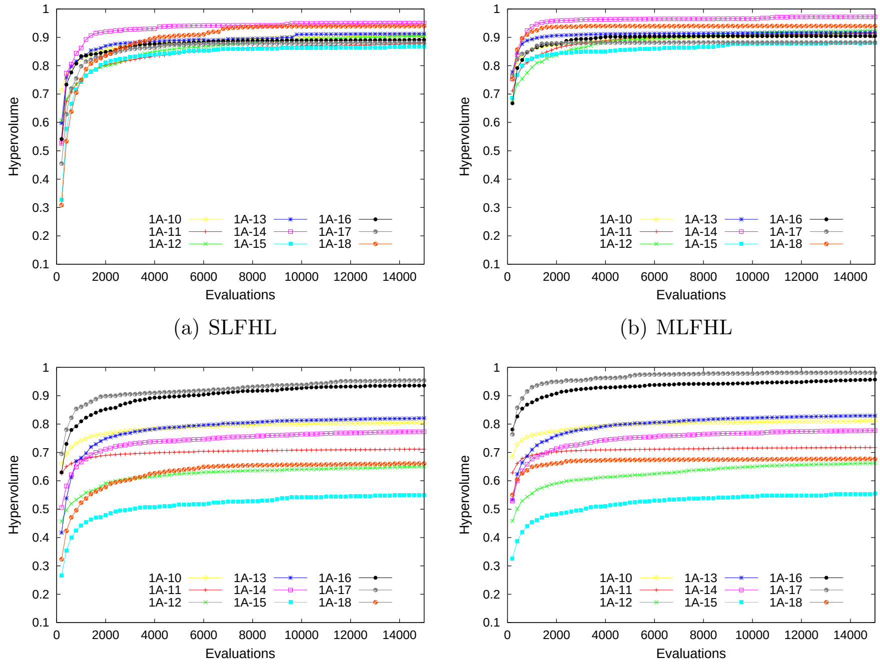


(d) MLFHD

Figure 6.16: Hypervolume for *1A* instances with $D = 2$

Instance	Different box types	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} $(Kg.)$
$1A-10$	5	$[5-10]$	10	5371	$6.15e^{07}$	$6.77e^{07}$
$1A-11$	5	$[5-10]$	20	2731	$6.15e^{07}$	$7.38e^{07}$
$1A-12$	5	$[5-10]$	30	5079	$6.15e^{07}$	$7.99e^{07}$
$1A-13$	5	$[15-20]$	10	380	$6.15e^{07}$	$6.77e^{07}$
$1A-14$	5	$[15-20]$	20	413	$6.15e^{07}$	$7.38e^{07}$
$1A-15$	5	$[15-20]$	30	750	$6.15e^{07}$	$7.99e^{07}$
$1A-16$	5	$[25-30]$	10	180	$6.15e^{07}$	$6.77e^{07}$
$1A-17$	5	$[25-30]$	20	180	$6.15e^{07}$	$7.38e^{07}$
$1A-18$	5	$[25-30]$	30	180	$6.15e^{07}$	$7.99e^{07}$

Table 6.36: *1A* problem instances with $D = 5$



(a) SLFHL (b) MLFHL

(c) SLFHD (d) MLFHD

Figure 6.17: Hypervolume for *1A* instances with $D = 5$

in depth, meaning that there is more free space left inside the container than when heuristics in length are used, as shown in Figure 6.5. But it could also be due to some relationship that exists between the ways in which the cuts and levels are generated in the different cutting and filling heuristics, respectively. Considering that the *1A* instances were generated using the 3D-MGL cutting heuristic, and that the best behaviour was achieved with SLFHL and MLFHL, it is possible that this could have an influence, since both filling heuristics apply the same process for the levels as the cutting heuristic for the cuts. Although evaluating the behaviour of each instance when executed using the different filling heuristics is made complex by the existence of multiple parameters that could affect their behaviour.

After concluding that SLFHL and MLFHL are the filling heuristics that yield the best quality results, it would be interesting to determine if there is some significant difference between them. Thus, and in an effort to ascertain if there is some difference that can be used to detect which yields better results, we conducted a

HEURISTICS	MLFHL								
	1A-10	1A-11	1A-12	1A-13	1A-14	1A-15	1A-16	1A-17	1A-18
SLFHL	$\leftrightarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\leftrightarrow$

Table 6.37: Statistical comparison between SLFHL and MLFHL with $D = 5$

Instance	Different box types	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} $(Kg.)$
$1A-19$	10	$[5-10]$	10	5397	$6.15e^{07}$	$6.77e^{07}$
$1A-20$	10	$[5-10]$	20	7569	$6.15e^{07}$	$7.38e^{07}$
$1A-21$	10	$[5-10]$	30	4313	$6.15e^{07}$	$7.99e^{07}$
$1A-22$	10	$[15-20]$	10	1184	$6.15e^{07}$	$6.77e^{07}$
$1A-23$	10	$[15-20]$	20	885	$6.15e^{07}$	$7.38e^{07}$
$1A-24$	10	$[15-20]$	30	717	$6.15e^{07}$	$7.99e^{07}$
$1A-25$	10	$[25-30]$	10	216	$6.15e^{07}$	$6.77e^{07}$
$1A-26$	10	$[25-30]$	20	216	$6.15e^{07}$	$7.38e^{07}$
$1A-27$	10	$[25-30]$	30	216	$6.15e^{07}$	$7.99e^{07}$

Table 6.38: *1A* problem instances with $D = 10$

statistical analysis. First we apply the Kolmogorov-Smirnov test in order to check whether the values of the results follow a normal (Gaussian) distribution or not. If so, the Levene test checks for the homogeneity of the variances. If samples have equal variance, an ANOVA test is done; otherwise, a Welch test is performed. For non-Gaussian distributions, the non-parametric Kruskal-Wallis test is used to compare the medians of the algorithms. A confidence level of 95% is considered, which means that the differences are unlikely to have occurred by chance with a probability of 95%. The analysis is performed using the hypervolume metric [231]. Table 6.37 shows the statistical comparison between SLFHL heuristic and the MLFHL heuristics with the *1A* instances. The study was carried out by comparing the statistical behaviour of both heuristics when used to the same instance. The symbol $\uparrow$ is used to denote that differences between the models are statistically significant and that the model in the left column yields a higher median and mean value. In the cases in which the opposite occurs, the symbol $\downarrow$ is used. Finally, for those cases in which the differences were not statistically significant, the symbol $\leftrightarrow$ is used. Thus, this analysis shows the high capacity of the MLFHL heuristic to achieve good results. Based on this study, we can conclude that MLFHL behaves better than SLFHL when evaluating instances with $D = 5$.

The next study considers the set of *1A* instances when $D = 10$. The Table 6.38 shows the set of instances and parameters used to generate them. Figure 6.18 shows the hypervolume obtained for each filling heuristic when using instances with the

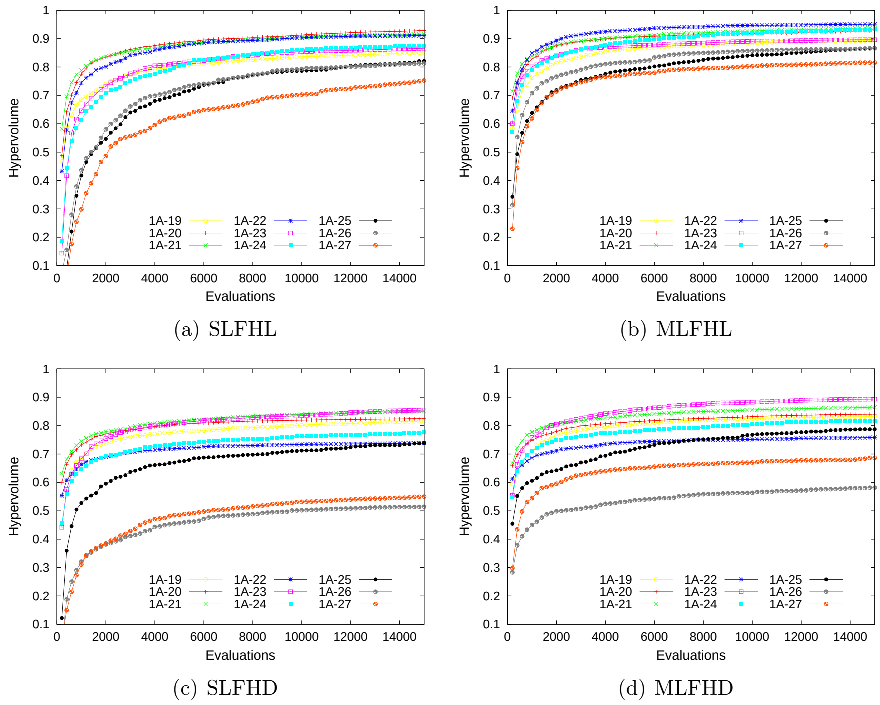


Figure 6.18: Hypervolume for *1A* instances with $D = 10$

input parameter $D = 10$. Based on Figure 6.18, SLFHL and MLFHL are the filling heuristics that manage to yield the best results for each instance type. Visually, however, it is difficult to ascertain if there are any significant differences between them that can be used to determine which one yields the best quality results for the set of instances in question. As a result, we conducted a statistical test in which we compared the two best filling heuristics detected for said instances ($D = 10$). Table 6.39 shows the statistical comparison between the different filling heuristics (SLFHL, MLFHL) when used to the same instance. The symbol $\uparrow$ is used to denote that differences between the models are statistically significant and that the model in the left column yields a higher median and mean value. In the cases in which the opposite occurs, the symbol $\downarrow$ is used. In every case, the good behaviour of MLFHL is evident. We can thus conclude that the filling heuristic that offers the best results

Heuristics	MLFHL								
	1A-19	1A-20	1A-21	1A-22	1A-23	1A-24	1A-25	1A-26	1A-27
SLFHL	$\downarrow$	$\leftrightarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$

Table 6.39: Statistical comparison between SLFHL and MLFHL with $D = 10$

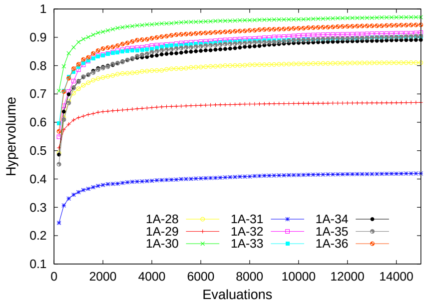
Instance	Different box types	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} $(Kg.)$
$1A - 28$	15	$[5 - 10]$	10	7533	$6.15e^{07}$	$6.77e^{07}$
$1A - 29$	15	$[5 - 10]$	20	9416	$6.15e^{07}$	$7.38e^{07}$
$1A - 30$	15	$[5 - 10]$	30	5473	$6.15e^{07}$	$7.99e^{07}$
$1A - 31$	15	$[15 - 20]$	10	781	$6.15e^{07}$	$6.77e^{07}$
$1A - 32$	15	$[15 - 20]$	20	837	$6.15e^{07}$	$7.38e^{07}$
$1A - 33$	15	$[15 - 20]$	30	762	$6.15e^{07}$	$7.99e^{07}$
$1A - 34$	15	$[25 - 30]$	10	405	$6.15e^{07}$	$6.77e^{07}$
$1A - 35$	15	$[25 - 30]$	20	405	$6.15e^{07}$	$7.38e^{07}$
$1A - 36$	15	$[25 - 30]$	30	405	$6.15e^{07}$	$7.99e^{07}$

Table 6.40: *1A* problem instances with $D = 15$

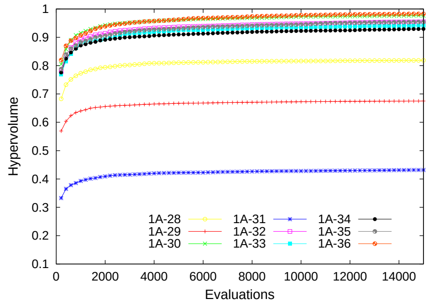
is MLFHL.

Finally, we evaluated the instances with input parameter $D = 15$. Table 6.40 shows the set of instances and the parameters used to generate them. Figure 6.19 shows the results for the hypervolume obtained by the set of filling heuristics with *1A* instances with the input parameter $D = 15$. In this case, all four filling heuristics: SLFHL, MLFHL, SLFHD and MLFHD output good quality results, though the best results are clearly given by MLFHL, with some solutions near the optimal. Although the good behaviour of the MLFHL filling heuristic is evident, it would be interesting to conduct a statistical test to compare SLFHL and MLFHL, since these are the filling heuristics that returned the best quality solutions. Table 6.41 shows this comparison. The symbol $\uparrow$ is used to denote that differences between the models are statistically significant and that the model in the left column yields a higher median and mean value. In the cases in which the opposite occurs, the symbol $\downarrow$ is used. This comparison study shows the good statistical behaviour that results when using MLFHL.

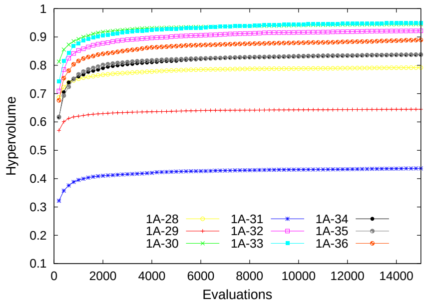
The set of experiments conducted on the *1A* problem instances determined that the filling heuristic that yields the best results is generally the MLFHL heuristic. However, we cannot ignore the possible relationship that might exist between the cutting heuristics used by 3D-MG and the filling heuristics employed to evaluate the solutions when both use the same procedure to create levels inside the container.



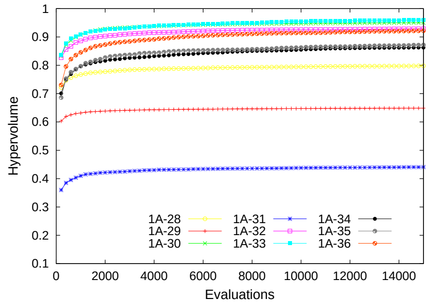
(a) SLFHL



(b) MLFHL



(c) SLFHD



(d) MLFHD

Figure 6.19: Hypervolume for *1A* instances with $D = 15$

Heuristics	MLFHL								
	1A-28	1A-29	1A-30	1A-31	1A-32	1A-33	1A-34	1A-35	1A-36
SLFHL	↓	↓	↔	↓	↓	↓	↓	↓	↓

Table 6.41: Statistical comparison between SLFHL, SLFHD, MLFHD and MLFHL with $D = 15$

6.5.3.2 Evaluation of the Input Parameters

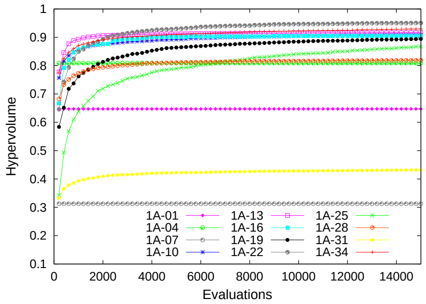
Based on the preceding studies, we determined that the filling heuristic that is generally able to offer quality solutions for most of the instances considered is the MLFHL heuristic. The remaining experiments will be based on the results obtained when executing the different instances with the MLFHL heuristic using NSGA-II. The experiments proposed attempt to determine what input parameter configuration for

3D-MG Input Parameters	Tuning values
$L \times W \times H$	$1200 \times 233 \times 220$
D	$2, 5, 10$, and 15
D_s	60%
$D_l - D_h$	$[5-10]\%, [15-20]\%$, and $[25-30]\%$
S_L	8.9%
W_G	$10\%, 20\%$, and 30%
$W_p[]$	$15.0\%, 25.0\%, 45.0\%, 55.0\%$

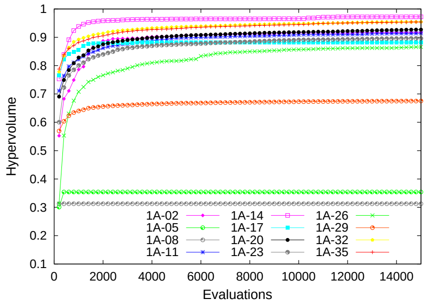
Table 6.42: Parameter values for generating the *1A* Instances

3D-MG will generate problems with different complexities when using the 3D-MGL cutting heuristic and the MLFHL heuristic. Table 6.42 shows the set of input parameters used to generate the *1A* problem instances. As we can see, parameters $L \times W \times H$, S_L and $W_p[]$ are constant. Therefore, the comparisons were made using the variable input parameters. We also hope to determine if the variation in these parameters has an effect on the complexity of the instances.

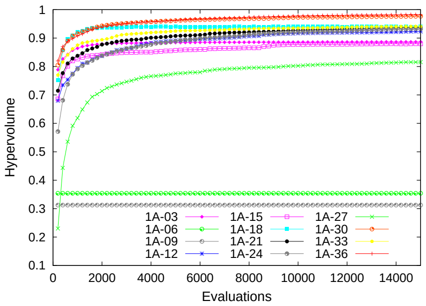
The first study we consider attempts to determine the value of the parameter W_G (total weight gain of Sol_2) that obtains quality solutions for the majority of the *1A* instances when these are executed using the MLFHL heuristic. W_G represents the increase in weight that takes place for the optimal-weight solution (Sol_2), which corresponds to the container's maximum (tare) weight, P_{max}. Thus, the higher the value of this parameter, the greater the degree of contradiction between the two objectives evaluated for the 3DPP studied in this thesis, namely the volume and weight. This is because it allows loading more weight inside the container while using the same volume inside the container. For the instances considered, this input parameter can have the values $W_G = 10, 20$, and 30. Figure 6.20 shows the hypervolume reached by MLFHL when applied to the set of instances with the different settings for the input parameter W_G. To simplify the experiment, we separated the instances according to the value assigned to W_G. Thus, the set of instances generated with $W_G = 10$ are all drawn together, as are the instances with $W_G = 20$ and $W_G = 30$. This makes it possible to see how using values of $W_G = 20$ and $W_G = 30$, for the generation of *1A* instances, the MLFHL yields fairly good quality solutions. For $W_G = 10$ with the MLFHL heuristic, the results are also good, though of lesser quality in some cases. This behaviour stems from the fact that with higher values of W_G, it is possible to fill the container with more weight, and thus obtain better quality solutions for the two objectives considered. This is because the search space is larger and the solutions are more scattered, which



(a) $W_G = 10$



(b) $W_G = 20$



(c) $W_G = 30$

Figure 6.20: Hypervolume for different values of W_G

allows finding better quality solutions. This might lead one to think that the best quality solutions should be obtained by executing MLFHL with the set of instances with the input parameter $W_G = 30$, since the container would be able to store a higher weight. If we compare Figure 6.20-b and Figure 6.20-c, we could conclude that the value of W_G that guarantees the highest quality solutions for MLFHL is 30, since for most of the instances evaluated, this value yields solutions very close to the optimal (above 0.9). However, it is not easy to visually evaluate which of the two values yields better quality solutions. As a result, it would be interesting to conduct a statistical analysis that can help us identify any significant differences that will allow us to determine the value for this input parameter, W_G, which allow to generate instances with different complexities. We did so by showing the robustness of the input parameter for different values. Table 6.43 shows the statistical comparison between the results obtained by MLFHL with the set of instances where the only parameter that varies is W_G. Any other comparison would be meaningless, since

WG	$W_G = 20/W_G = 30$		
$D = 5$	$1A - 11/1A - 12$	$1A - 14/1A - 15$	$1A - 17/1A - 18$
	$\leftrightarrow$	$\uparrow$	$\downarrow$
$D = 10$	$1A - 20/1A - 21$	$1A - 23/1A - 24$	$1A - 26/1A - 27$
	$\leftrightarrow$	$\downarrow$	$\uparrow$
$D = 15$	$1A - 29/1A - 30$	$1A - 32/1A - 33$	$1A - 35/1A - 36$
	$\downarrow$	$\uparrow$	$\downarrow$

Table 6.43: Statistical comparison between $W_G = 20$ and $W_G = 30$

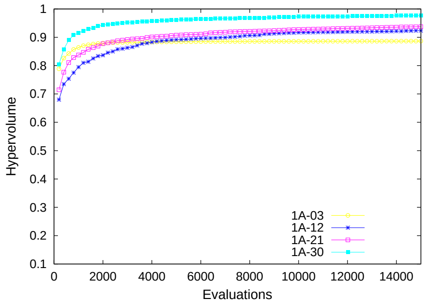
the instances were generated using very different configurations. The symbol $\uparrow$ is used to denote that differences between the models are statistically significant and that the model more to the left ($W_G = X/W_G = Y$) yields a higher median and mean value. In the cases in which the opposite occurs, the symbol $\downarrow$ is used. In this study we did not include the results from instances with $D = 2$, since they are not statistically comparable to the rest. Note how the MLFHL heuristic exhibit very good statistical behaviour when using both values of W_G.

From the above study we conclude that the instances generate with the value of $W_G = 30$ are more easier to solve by MLFHL heuristic. As a result, the remaining experiments will be conducted with this value for said input parameter. In the following experiment, we will evaluate the behaviour of input parameters D_l and D_h, used by 3D-MG to generate the *1A* benchmark. Input parameters D_l and D_h determine the lowest and highest dimensions that a box type can have with respect to the dimensions of the container, respectively. Table 6.25 shows the possible values that parameters D_l, D_h can take, these being $[D_l - D_h] = [5 - 10]\%$, $[D_l - D_h] = [15 - 20]\%$ and $[D_l - D_h] = [25 - 30]\%$. These parameters are studied as if they were a single parameter. Combining the values of D_l and D_h is not possible for the implementation carried out for 3D-MG. A set of boxes is considered to be small when its dimensions are between $[5-10]\%$ of the length, height and width of the container, and large when its dimensions are between $[25 - 30]\%$. Boxes outside these ranges are regarded as medium sized. The set of instances evaluated in this experiment is shown in Table 6.44. The hypervolume [231] obtained for MLFHL heuristic for each instance is shown in Figure 6.21. This one shows the hypervolume obtained by the filling heuristic with various instances with different values for the input parameters D_l and D_h. So as to simplify the evaluation of these input parameters, the instances evaluated in this experiment were divided based on D_l and D_h. Thus, all of the instances with the same value of D_l and D_h were drawn together. For instances *1A-03*, *1A-06* and *1A-09*, corresponding to the instances with $D = 2$ and $W_G = 30$ as the variable parameters, we see that the results are not competitive. This could be because the instances are highly complex and the MLFHL cannot process them

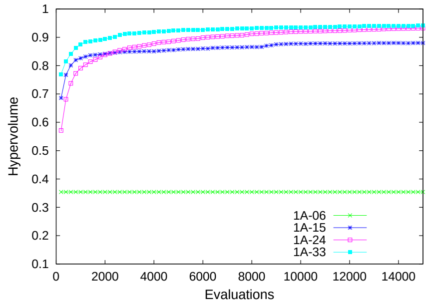
Instance	Different box types	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} $(Kg.)$
$1A-03$	2	$[5-10]$	30	4394	$6.15e^{07}$	$7.99e^{07}$
$1A-06$	2	$[15-20]$	30	250	$6.15e^{07}$	$7.99e^{07}$
$1A-09$	2	$[25-30]$	30	54	$6.15e^{07}$	$7.99e^{07}$
$1A-12$	5	$[5-10]$	30	5079	$6.15e^{07}$	$7.99e^{07}$
$1A-15$	5	$[15-20]$	30	750	$6.15e^{07}$	$7.99e^{07}$
$1A-18$	5	$[25-30]$	30	180	$6.15e^{07}$	$7.99e^{07}$
$1A-21$	10	$[5-10]$	30	4313	$6.15e^{07}$	$7.99e^{07}$
$1A-24$	10	$[15-20]$	30	717	$6.15e^{07}$	$7.99e^{07}$
$1A-27$	10	$[25-30]$	30	216	$6.15e^{07}$	$7.99e^{07}$
$1A-30$	15	$[5-10]$	30	5473	$6.15e^{07}$	$7.99e^{07}$
$1A-33$	15	$[15-20]$	30	762	$6.15e^{07}$	$7.99e^{07}$
$1A-36$	15	$[25-30]$	30	405	$6.15e^{07}$	$7.99e^{07}$

Table 6.44: *1A* problem instances with $W_G = 30$

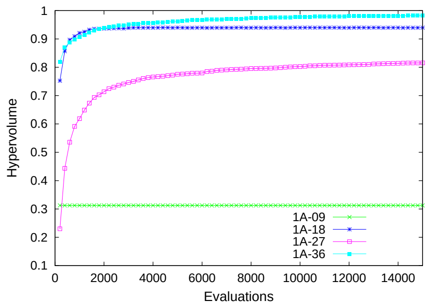
properly, and is thus unable to find a feasible packing pattern. For the remaining instances where D is still a variable parameter, we see how the MLFHL yield good quality solutions, often approaching 0.9, very close to the optimal solutions. Note, however, how the filling heuristic yield promising results when we use the instances with values $[D_l - D_h] = [5-10]\%$ and $[D_l - D_h] = [15-20]\%$. There is a certain improvement when $[D_l - D_h] = [5-10]\%$ is used. In order to provide results with statistical confidence, suitable statistical analyses must be performed. The statistical analysis for MLFHL with the instances with $D = 2$ are not included because they are not statistically comparable. Table 6.45, Table 6.46 and Table 6.47 show whether the row configuration is statistically better (↑), not different (↔), or worse (↓) than the corresponding column combination. Table 6.45 shows the statistical analysis when $D = 5$ and $W_G = 30$ so that we can compare the instances in which the variable value corresponds to the input parameters $[D_l - D_h]$. Any other comparison is not feasible since the instances were generated using different input parameters. The same statistical comparison is shown in Table 6.46, in this case for $D = 10$ and $W_G = 30$. And finally, Table 6.47 shows a comparison between the results obtained by MLFHL when using the instances with $D = 15$ and $W_G = 30$. Based on the statistical results, it is not possible to determine which of the values assigned to $[D_l - D_h]$ for the selected filling heuristic will guarantee the best quality results, since they depend on input parameter D. In other words, that the problem instances are more or less complex depends on the parameter D. For $D = 5$, MLFHL achieved the best value when $[D_l - D_h] = [25-30]\%$, while for $D = 10$ there are no significant differences between $[D_l - D_h] = [5-10]\%$ and $[D_l - D_h] = [15-20]\%$. Finally, for $D = 15$ we determined that MLFHL obtained the best values with



(a) $D_l - D_h = 5 - 10$



(b) $D_l - D_h = 15 - 20$



(c) $D_l - D_h = 15 - 30$

Figure 6.21: Comparison between different box sizes with $W_G = 30$

	$[5-10]$ 1C-12	$[15-20]$ 1C-15	$[25-30]$ 1C-18
$[5-10]$	$\leftrightarrow$	$\uparrow$	$\downarrow$
$[15-20]$	$\downarrow$	$\leftrightarrow$	$\downarrow$
$[25-30]$	$\uparrow$	$\uparrow$	$\leftrightarrow$

Table 6.45: Statistical comparison for $[D_l - D_h]\%$ with $D = 5$ and $W_G = 30$

$[D_l - D_h] = [5 - 10]\%$ and $[D_l - D_h] = [25 - 30]\%$. For these cases, they are simple instances of solving for our filling heuristic.

Having evaluated the input parameters WG and $[D_l - D_h]$, we now focus on D, the last variable parameter used to generate the *1A* problem instances. However, in the preceding study we concluded that the value of $[D_l - D_h]$ that allows to generate easier instances to solve by MLFHL depends on D. Therefore, conducting

	$[5-10]$ 1C-21	$[15-20]$ 1C-24	$[25-30]$ 1C-27
$[5-10]$	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$
$[15-20]$	$\leftrightarrow$	$\leftrightarrow$	$\uparrow$
$[25-30]$	$\downarrow$	$\leftrightarrow$	$\leftrightarrow$

Table 6.46: Statistical comparison for $[D_l - D_h]\%$ with $D = 10$ and $W_G = 30$

	$[5-10]$ 1C-30	$[15-20]$ 1C-33	$[25-30]$ 1C-36
$[5-10]$	$\leftrightarrow$	$\uparrow$	$\leftrightarrow$
$[15-20]$	$\downarrow$	$\leftrightarrow$	$\downarrow$
$[25-30]$	$\leftrightarrow$	$\uparrow$	$\leftrightarrow$

Table 6.47: Statistical comparison for $[D_l - D_h]\%$ with $D = 15$ and $W_G = 30$

a study to determine the value of D that allows MLFHL to achieve quality solutions is meaningless since we would be ruling out good solutions that stem from applying the same value of WG, but different values of $[D_l - D_h]$ and D. Thus, in the case of the *1A* instances, we omitted the study on D. The possible values that D can take are shown in Table 6.25, and correspond to $D = 2, 5, 10, 15$.

The study involving the input parameters was complex since, as we were able to conclude, there is no combination of input parameters that guarantees that quality solutions can always be obtained by the MLFHL heuristic for the *1A* instances. The complexity of the instances are highly dependent on the input parameters. We were, however, able to determine that the larger the W_G parameter, the better the quality of the results using MLFHL. Given the relationship between this parameter and the container's maximum weight, P_{max}, we can conclude that there might exist a better balance between the two objectives studied for the 3DPP.

Part III

Conclusions and Future Work

Chapter

7

Conclusions

This chapter provides a review of the work carried out to study the Three-Dimensional Packing Problem (3DPP) as an intrinsically multi-objective problem, solved by employing multi-objective techniques. We thus present the set of conclusions that we have reached after using computational evaluations to validate the efficiency of the proposals developed in this thesis.

7.1 Conclusions

Many engineering and logistics problems applied to industry implicitly work with more than one objective. When they are studied in the literature, however, many of these problems tend to be formulated as mono-objective problems. The 3DPP is an inherently multi-objective problem whose goal is to efficiently arrange a set of boxes inside one or more containers. In most cases, the sole objective considered is the volume, ignoring other important factors, such as the weight. Standard containers are defined using a series of parameters that include the maximum cargo volume and the weight, or tare, limit. These two elements are essential when selecting any type of container. This is what we wanted to focus on in this work, by studying a directly multi-objective formulation of the 3DPP where the volume and weight are considered as two objectives to be maximised simultaneously.

There are isolated papers that address the mono-objective solution of the 3DPP using exact algorithms. However, dealing with the multi-objective variant of this problem through exact algorithms would overly complicate its resolution. The 3DPP is classified as an NP-hard problem, the complexity of which depends not only on the large number of possible locations available for the boxes, but also on the multiple restrictions that may apply, such as the loading priorities or the orientation of the boxes. Thus, dealing with the multi-objective formulation of this problem in a reasonable time frame using an exact method is only feasible for very small instances that are far removed from the reality that affects this type of problem. Because of this, in this work we have focused on other methods that can allow us to solve the multi-objective version of the problem in a reasonable amount of time. Specifically, MOEAs have shown promise in solving this kind of formulation. As a result, we have applied them to our multi-objective formulation of the 3DPP.

When working with evolutionary algorithms, we have to define a set of specifications that are intrinsic to the problem type, such as: coding the population, the various operators and the evaluation function. In the case of the multi-objective 3DPP studied in this paper, we used a coding scheme that specifies the sequence for placing and orienting the boxes. This scheme has been shown to be promising for defining our problem. Numerous variation operators were tested in an effort to find those that introduced the most variation within the population. The mutation operators studied include the add one gene mutation, the remove one gene mutation, the change a gene mutation, the modify some genes mutation, the swap two genes mutation, and the shift a gene mutation. Of these, the ones that provided the best quality results were the add one gene mutation, the remove one gene mutation, and the change a gene mutation. The remaining mutation operators did not improve on the solutions obtained previously by the selected operators (add, remove, change), and were thus omitted from subsequent studies. The implemented crossover operator was the one-point crossover, based on the specifications of our problem where, for example, it must hold that at the end of the crossover process, there cannot be more pieces than those defined for a given piece type. Finally, we implemented various filling heuristics based on construction approaches, such as the wall-building approach and the vertical-layer-building approach. The set of filling heuristics developed rely on the information encoded in each of the genes that make up the chromosome for arranging pieces inside the container. A constraint considered in the filling heuristics was the orientation, with the boxes being allowed to be rotated about their base. With these premises, and taking into account some assumptions, we designed different evaluation functions for the purpose of finding a loading pattern that offered the best solution in terms of both volume and weight. These filling heuristics are essential for evaluating the solutions that represent the candidate solutions. In this work, we have presented four filling heuristics:

- Single-Level Filling Heuristic in Length (SLFHL)
- Multiple-Level Filling Heuristic in Length (MLFHL)
- Single-Level Filling Heuristic in Depth (SLFHD)

- Multiple-Level Filling Heuristic in Depth (MLFHD)

The SLFHL and the MLFHL were implemented considering a wall-building approach, while in SLFHD and MLFHD, we considered a vertical-layer-build approach. It was interesting to observe the performance of the MOEAs when the solutions were evaluated using filling heuristics whose levels are open in terms of length and depth. The SLFHL and SLFHD are filling heuristics in which only one level can be open in each iteration. Although one might initially think that the logical way to load a container is to use constructive techniques, like the employed in SLFHD, experiments have shown that this practice does not always yield good results. And so, contrary to what one might think beforehand, we noticed that the best results were obtained when applying SLFHL, in which if a piece cannot be placed in the level that is currently open, it is ignored and the next piece is considered. Although both heuristics follow an order, it is not strict, and the sequence given in the chromosome is lost as a result of the population's evolutionary process. It is in this scenario that the filling heuristics MLFHL and MLFHD arise. In contrast to SLFHL and SLFHD, the modified heuristics MLFHL and MLFHD manage a dynamic set of layers in which to place a given item. Following our studies, we concluded that maintaining the order encoded in the chromosome for placing items inside the container is important, as this has yielded the highest quality solutions.

We also verified the importance of using frameworks or tools like METCO [142], which generally provide the global and internal characteristics of the algorithms, allowing the user to focus solely on the most specific aspects of the problem. The user only has to develop the associated plugin in C++ to define the individual that represents the problem. In order to define the individuals that will form the population, METCO provides a series of methods that the user must redefine to adapt them to the specific problem.

Although several benchmark libraries exist that contain cutting and packing problem instances, we were unable to find any instances that could be applied to the multi-objective formulation of the 3DPP proposed in this paper. The instances that refer to the 3DPP only consider the volume as their objective, ignoring the weight of both the container and the boxes. We only found one real instance that could be applied to our problem's specifications. This instance was referenced and used by Dereli et al. in [70]. It is an instance from a distribution company that considers 12 different types of products that have to be arranged inside a vehicle of given dimensions and tare. The volume and weight of each piece are specified. The instance did not consider the rotation of the pieces, however, and thus the decision was made that the pieces could only be rotated about their bases, since many of them contained liquids. But conducting a computational study based on a single instance is not conclusive, since the results rely on a single experiment. Another possible option for studying the formulation is to contact real companies, but this is not always easy, since we often received no reply. Motivated by this gap in the literature, and given the essential nature of having a set of instances to work with and evaluate the group of proposals developed, we devised software that would allow us to generate different types of instances. Thus we developed the 3D-MG, an instance generator for the multi-objective formulation of the 3DPP. Based on the two types of filling heuristics (in length and in depth) developed to decode the solutions for the evolutionary algorithm, we generated two approaches for generating the different instances:

- Three-Dimensional Multi-Objective Generator in Length (3D-MGL)
- Three-Dimensional Multi-Objective Generator in Depth (3D-MGD)

In the 3D-MGL cutting heuristic, the cuts made inside the container follow the same procedure as the SLFHL and MLFHL filling heuristics when they create the different levels inside the container. Thus, the 3D-MGD cutting heuristic generates cuts in the same way as the SLFHD and MLFHD filling heuristics when they create the various levels. In this work we provide to the literature an instance generator that can be used to generate instances 3DPP in which the volume and weight are evaluated simultaneously. Although several sets of instances were created using the 3D-MG, only a few are presented in this work; specifically, a total of 72 instances were created using the 3D-MGL cutting heuristic, resulting in 36 instances of the so-called *1C* problem instances, and another 36 of the type *1A* problem instances, in keeping with the container names employed in the ISO (International Organization for Standardization) standard.

Other sets of experiments were conducted to validate the input parameters used to generate the various instances. Based on these, we concluded that using high values for the W_G input parameter allows increasing the degree of conflict between the two objectives. This is because the value of W_G is crucial to calculating the weight of the container, and results in the difference between the container's weight and its volume being more noticeable. We also verified that the *1C* and *1A* instances that attain the best load distributions are those that were generated based on the values $W_G = 30$ and $[D_l - D_h] = [15 - 20]$. These correspond to medium-sized boxes. This is possibly because the amount of free space that is spread out inside the container when larger or smaller boxes are considered is more prominent.

The loading patterns are the result of executing different MOEAs configurations applied to a specific instance. These results are generally output to text files that might not be understood by an end user, since they are formatted to be processed computationally. So, in an effort to find a display tool that can be used to visualize the loading pattern generated, we conducted an online bibliographic review of display tools. After finding no applications that met our requirements, we considered developing our own display tool, thus adding another powerful work tool to the literature. Using this display tool, we were able to evaluate the performance of each heuristic by applying them to different instances. This also turned the tool into a powerful source for identifying improvements.

So as to validate the performance of MOEAs when applied to the multi-objective formulation of the 3DPP, we selected three of the best-known MOEAs in the literature:

- Non-Dominated Sorting Genetic Algorithm II (NSGA-II)
- Strength Pareto Evolutionary Algorithm 2 (SPEA2)
- Adaptive Indicator-Based Evolutionary Algorithm (Adaptive-IBEA)

In an effort to be able to do a comparison that would allow us to validate the multi-objective proposed in this thesis, we ran the different filling heuristics on the only real instance known in the literature, used in Dereli et al. in [70]. A series of computational experiments revealed the good performance of the MOEAs when applied to the 3DPP. When the different filling heuristics were executed with MOEAs, such as the NSGA-II and the SPEA2, we were able to improve the results obtained in every case by Dereli et al. by using a Simulated Annealing (SA) with a filling heuristic. Even the Adaptive-IBEA, which yielded the lowest quality results for our proposals, managed to outperform their results. We can thus conclude that using specific techniques for solving the 3DPPs as a multi-objective problem yielded better results than those in [70]. It is apparent that applying multi-objective formulations to this class of problem is advantageous, not only because it better reflects the reality in the industry, but because it yields high-quality results that are even superior to those already known. Our study also allowed us to validate the great effectiveness of MOEAs when applied to these kinds of real-world problems.

To validate the performance of each of the filling heuristics proposed in this work, we conducted several experiments using different sets of instances, which allowed us to draw certain conclusions. For one thing, we were able to determine that for the set of benchmarks studied, the filling heuristics that provide the highest quality results are SLFHL and MLFHL. SLFHD and MLFHD, despite also offering good results, are not as good as SLFHL and MLFHL. In this regard, we also considered the possible existence of a relationship between the filling heuristics that returned the best quality solutions (SLFHL, MLFHL) and the cutting heuristic applied to generate the set of *1C* and *1A* problem instances (3D-MGL). We were unable to draw any clear conclusions, however, since the same result was obtained when the computational tests were conducted with the real instance from the company that distributed Procter & Gamble's products [70]. Multiple comparisons carried out based on SLFHL and MLFHL demonstrated that the filling heuristic that yields the best quality solutions for the set of instances studied is the MLFHL heuristic.

Regarding the parallel approaches, several island models were used in order to see if parallel approaches could be used to improve the results obtained using sequential models or to reduce the amount of time needed to achieve results of at least comparable quality. This study was carried out using the SLFHL filling heuristic, the goal being to improve the solutions obtained. The resulting study revealed that all of the homogeneous parallelizations introduced to solve the 3DPPs improved on the sequential results, yielding solutions of equal quality in fewer evaluations, and thus in less time. Some parallelizations are even able to improve the quality of sequential solutions, as in the case of the two-island execution. The eight-island parallel model, however, is not able to replicate the final quality obtained by the sequential proposal in the same number of evaluations, since when the number of islands is increased, the number of evaluations is spread out among more processors, meaning each island advances less in its exploration. In summary, we conclude that the use of multi-objective techniques improved the results presented in the only reference work found in the literature that considers the weight as second objetive.

Chapter 8

Future Work

Over the course of this work, we have proposed a series of improvements that we would like to outline in this chapter as future lines of research.

8.1 Future Work

There are several open lines of research that can be studied based on the multi-objective definition of the Three-Dimensional Packing Problem (3DPP) proposed in this work. On the one hand, and given the good quality of the results obtained when using MOEAs to solve the 3DPP, it would be interesting to try the set of filling heuristics and validate them using not only the MOEAs, but other algorithms of interest. In this regard, these algorithms could be tested and compared so as to evaluate their performance.

Many of the studies found in the literature consider the 3DPP without taking into account the restrictions normally encountered in real life. As a result, it is vitally important that more realistic restrictions be applied to the heuristics proposed in this work. One interesting restriction that could be applied to the filling heuristics presented is the stacking control for the freight inside the container, in that heavy boxes cannot be placed atop lighter ones. Another easily incorporated restriction that is useful to transport logistics is the loading priorities; that is, establishing an order for the industrial processes of loading and unloading freight. These are key elements if the research is to be marketed to industry. The Canary Islands are one potential market of application, given the high cargo traffic both at sea and on land [182]. There are many other significant practical restrictions, however, that have to be considered due to their relevance to the real world. As a result of the foregoing, including more restrictions in the study will be an important area of focus in future expansions to this work.

On the other hand, it would be beneficial to study more in detail if there is actually a relationship between the cutting heuristics and the filling heuristics, and if any such relationship influences the results. It would be advisable to generate instances using the 3D-MGD cutting heuristic and execute them with the set of filling heuristics, in effect conducting an in-depth study of the quality of the solutions when cutting and filling heuristics are used and comparing them to the filling heuristics in length when using the instances created with the 3D-MGD. This would serve to check whether a relationship exists or not. If it does, one future area of work would be to perfect the set of filling heuristics for any type of instance.

Another interesting line of work could be the generation of more types of instances. It would be useful to develop other proposals for generating instances that could take into account other kinds of restrictions, such as the handling of non-guillotinable cuts and even of irregular pieces. These changes would entail alterations to the filling heuristics, but their processing would nevertheless be interesting on a computational level.

Another area of research that remains open is the use of applied parallel models when using the four approaches developed for evaluating the individuals in the population. This possibility exists thanks to the METCO tool, which can execute different parallel models, both homogeneous and heterogeneous, not only for the purpose of saving time by spreading the computational effort, but to obtain the algorithmic benefits derived from cooperation among several populations, as happens in heterogeneous island models. This could be used to do comparisons when adding new restrictions, which would probably require a more intense computational effort. Based on this, their applicability to other studies that are conducted could be validated.

Due to the computational complexity exhibited by the 3DPP studied, another interesting area of future work would be to apply multi-objectivization to the 3DPP. This could be done, for example, by weighting the volume and weight objectives to make the second objective fictitious so as to enable the application of multi-objective algorithms. Or the two objectives of volume and weight could be kept and a third, fictitious, objective added, such as minimising the distance between the solutions on the Pareto front.

Finally, it is of great interest to be able to validate a study by applying it to real cases. Still unfinished is the task of contacting companies devoted to transporting freight, or even pharmaceutical companies, which have also expressed certain interest in this research due to their use of medicine vending machines that might require efficient mechanisms for arranging and extracting products. Undertaking this challenge would give our research a more practical application by giving us first-hand knowledge of the most common restrictions in the industry and allowing us to adjust our heuristics to the real needs of companies.

Part IV

Appendice

Appendix

A

List of Publications

This appendix presents the publications that have been produced as a result of the research conducted over the course of this PhD. These publications include contributions to international conferences with review committees for ensuring the quality and validity of the selected works, as well as contributions to national conferences. Adding to this appendix, the works present in seminars of relevance.

International Conferences

[1] Gara Miranda, Algirdas Lančinskas, and Yanira González. Single and Multi-Objective Genetic Algorithms for the Container Loading Problem. In *Genetic and Evolutionary Computation Conference*. To appear, July 2017.

[2] Yanira González, Gara Miranda, and Coromoto León. An Instance Generator for the Multi-Objective 3D Packing Problem. In Manuel Graña, José Manuel López-Guede, Oier Etxaniz, Álvaro Herrero, Héctor Quintián, and Emilio Corchado, editors, *International Joint Conference SOCO'16-CISIS'16-ICEUTE'16, 2016 Proceedings*, pages 386–396. Springer International Publishing, October 2016.

[3] Yanira González, Gara Miranda, and Coromoto León. Multi-objective Multi-level Filling Evolutionary Algorithm for the 3D Cutting Stock Problem. In *Knowledge-Based and Intelligent Information and Engineering Systems: Proceedings of the 20th International Conference KES-2016*, volume 96, pages 355 – 364. Procedia Computer Science, 2016.

[4] Y. González, C.León, and G. Miranda. A Hyper-Heuristic Approach to solve the Multi-Objective Container Loading Problem. In *5th International Conference on Metaheuristics and Nature Inspired Computing (META'2014)*, October 2014.

[5] Yanira González, Coromoto León, Gara Miranda, and Javier Villamonte. Graphical User Interface for the Container Loading Problem: An approach using JavaScript. In Carina González, César Collazos, Habib Fardoun, Martín Llamas, Carlos Vaz, Pedro Latorre, and Inmaculada Perdomo, editors, *XV Congreso Internacional Interacción Persona-Ordenador. Actas*, pages 6 – 7. Asociación Interacción Persona-Ordenador (AIPO), 2014.

[6] Yanira González, Gara Miranda, and Coromoto León. A Multi-level Filling Heuristic for the Multi-objective Container Loading Problem. In *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13, 2013 Proceedings*, pages 11–20, September 2013.

[7] J. De Armas, Y. González, G. Miranda, and C. León. Parallelization of the Multi-Objective Container Loading Problem. In *2012 IEEE Congress on Evolutionary Computation, CEC 2012*, pages 155–162, June 2012.

[8] Y. González J. de Armas, G. Miranda, and C.León. Parallel Models for the Multi-Objective Container Loading Problem. In *9th ESICUP Meeting*, March 2012.

National Conferences

[1] J. de Armas, Y. González, G. Miranda, and C. León. El Problema de Carga de Contenedores: una Aproximación Paralela y Multi-objetivo. In *VIII Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'2012)*, pages 399–406, February 2012.

International Seminar

[1] Yanira González, Gara Miranda, and Coromoto León. Evaluation of placement heuristics for the Container Loading Problem. Seminar paper, Seminar in Latest Advances in Computer Science, November 2015.

[2] Yanira González, Jesica de Armas, Gara Miranda, and Coromoto León. A comparative Study of Evaluations Heuristics for Solving the Multi-Objective Container Loading Problem. Seminar paper, EURO Summer Institute on Cutting and Packing, July 2012.

Appendix

B Pseudo-Code

Algorithm 6 Single-Level Filling Heuristic in Length

pieces = $q_1, q_2, \dots, q_N$
freeWidth = W
totalVolume = 0
totalWeight = 0
$EMS_x = \emptyset$ EMS-front list
$EMS_y = \emptyset$ EMS-beside list
$EMS_z = \emptyset$ EMS-above list
while ($pieces \neq \emptyset$) and ($\exists q_x \subset pieces,\ q_x \in t_i, i \in [1, m]$ / $(freeWidth \geq w_i)$ y $((W - freeWidth < W)$ and $(totalWeight + p_i \leq P_{max})$)) **do**
    create a layer with width(layer) = w_i, length(layer) = L, height(layer) = H
    freeWidth = freeWidth - w_i
    place the piece inside the container
    totalVolume = totalVolume + v_i
    totalWeight = totalWeight + p_i
    if (lenght(layer) - $l_i \neq 0$) **then**
        create EMS in the y-axis and create the EMS in the EMS-beside list
        length(layer) = $L - l_i$
        width(layer) = w_i
        height(layer) = h_i
    end if
    if (height(layer) - $h_i \neq 0$) **then**
        create EMS in the z-axis and create the EMS in the EMS-above list
        length(layer) = L
        width(layer) = w_i
        height(layer) = $H - h_i$
    end if
    remove q_x from pieces
    while ($\exists q_x \subset pieces, q_x \in t_i, i \in [1, m]$ / $EMS \in EMS - beside$ or $EMS \in EMS - front$ or $EMS \in EMS - above, l_i \leq length(EMS), w_i \leq width(EMS), h_i \leq heigth(EMS)$) and $(totalWeight + p_i \leq P_{max})$ **do**
        if ($l_i == length(EMS)$) and ($w_i == widthh(EMS)$) and ($h_i == heigth(EMS)$) **then**
            remove EMS from the corresponding list
        else
            remove EMS from the corresponding list
            if ($width(EMS) - w_i \neq 0$) **then**
                create EMS in the x-axis and create the EMS in the EMS-front list
                length(EMS) = l_i
                width(EMS) = $width(EMS) - w_i$
                height(EMS) = h_i
            end if
            if ($length(EMS) - l_i \neq 0$) **then**
                create EMS in the y-axis and create EMS in the EMS-beside list
                length(EMS) = $length(EMS) - l_i$
                width(EMS) = $width(EMS)$
                height(EMS) = h_i
            end if
            if ($height(EMS) - h_i \neq 0$) **then**
                create EMS in the z-axis and create EMS in the EMS-above list
                length(EMS) = $length(EMS)$
                width(EMS) = $width(EMS)$
                height(EMS) = $height(EMS) - h_i$
            end if
        end if
        place the piece inside the container
        $totalVolume = totalVolume + v_i$
        $totalWeight = totalWeight + p_i$
        remove q_x from pieces
    end while
end while

Algorithm 7 Multiple-Level Filling Heuristic in Length

pieces = $q_1, q_2, \ldots, q_N$
freeWidth = W
totalVolume = 0
totalWeight = 0
$EMS_x = \emptyset$ EMS-front list
$EMS_y = \emptyset$ EMS-beside list
$EMS_z = \emptyset$ EMS-above list
while $(pieces \neq \emptyset)$ and $(\exists q_x \subset pieces, q_x \in t_i, i \in [1,m] \ / \ (totalWeight + p_i \leq P_{max})$ and $(totalVolume + v_i \leq (LxWxH))$ **do**
  while $(\exists q_x \subset pieces, q_x \in t_i, i \in [1,m] \ / \ EMS \in EMS-beside$ or $EMS \in EMS-front$ or $EMS \in EMS-above, l_i \leq length(EMS), w_i \leq width(EMS), h_i \leq heigth(EMS))$ and $(totalWeight + p_i \leq P_{max})$ **do**
    if $(l_i == length(EMS))$ and $(w_i == widthh(EMS))$ and $(h_i == heigth(EMS))$ **then**
      remove EMS from the corresponding list
    else
      remove EMS from the corresponding list
      if $(width(EMS) - w_i \neq 0)$ **then**
        create EMS in the x-axis and create EMS in the EMS-front list
        length(EMS) = l_i
        width(EMS) = $width(EMS) - w_i$
        height(EMS) = h_i
      end if
      if $(length(EMS) - l_i \neq 0)$ **then**
        create EMS in the y-axis and create EMS in the EMS-beside list
        length(EMS) = $length(EMS) - l_i$
        width(EMS) = $width(EMS)$
        height(EMS) = h_i
      end if
      if $(height(EMS) - h_i \neq 0)$ **then**
        create EMS in the z-axis and create in the EMS-above list
        length(EMS) = $length(EMS)$
        width(EMS) = $width(EMS)$
        height(EMS) = $height(EMS) - h_i$
      end if
    end if
    place the piece inside the container
    $totalVolume = totalVolume + v_i$
    $totalWeight = totalWeight + p_i$
    remove q_x from pieces
  end while
  if $(pieces \neq \emptyset)$ and $(\exists q_x \subset pieces, \ q_x \in t_i, i \in [1,m] \ / \ (freeWidth \geq w_i)$ and $(q_x \not\exists x_i, x_i \in [0, b_i], i \in [1,m]))$ **then**
    create a layer with width(layer) = w_i, length(layer) = L, height(layer) = H
    freeWidth = freeWidth - w_i
    place the piece inside the container
    totalVolume = totalVolume + v_i
    totalWeight = totalWeight + p_i
    if (lenght(layer) - $l_i \neq 0$) **then**
      create EMS in the y-axis and create EMS in the EMS-beside list
      length(layer) = $L - l_i$
      width(layer) = w_i
      height(layer) = h_i
    end if
    if (height(layer) - $h_i \neq 0$) **then**
      create EMS in the z-axis and create EMS in the EMS-above list
      length(layer) = L
      width(layer) = w_i
      height(layer) = $H - h_i$
    end if
    remove q_x from pieces
  end if
end while

Algorithm 8 Single-Level Filling Heuristic in Depth

```
pieces = q_1, q_2, ..., q_N
freeLength = W
totalVolume = 0
totalWeight = 0
EMS_x = ∅ EMS-beside list
EMS_y = ∅ EMS-front list
EMS_z = ∅ EMS-above list
while (pieces ≠ ∅) and (∃q_x ⊂ pieces, q_x ∈ t_i, i ∈ [1, m] / (freeLength ≥ w_i) y ((L − freeLength < L) and
(totalWeight + p_i ≤ P_max))) do
    create a level (wall) with width(wall) = W, length(wall) = l_i, height(wall) = H
    freeLength = freeLength - l_i
    place the piece inside the container
    totalVolume = totalVolume + v_i
    totalWeight = totalWeight + p_i
    if (width(layer) - w_i ≠ 0) then
        create EMS in the x-axis and create EMS in the EMS-front list
        length(wall) = l_i
        width(wall) = W − w_i
        height(wall) = h_i
    end if
    if (height(wall) - h_i ≠ 0) then
        create EMS in the z-axis and create EMS in the EMS-above list
        length(wall) = l_i
        width(wall) = W
        height(wall) = H − h_i
    end if
    remove q_x from pieces
    while (∃q_x ⊂ pieces, q_x ∈ t_i, i ∈ [1, m] / EMS ∈ EMS − beside or EMS ∈ EMS − front or EMS ∈
    EMS−above, l_i ≤ length(EMS), w_i ≤ width(EMS), h_i ≤ heigth(EMS)) and (totalWeight+p_i ≤ P_max)
    do
        if (l_i == length(EMS)) and (w_i == widthh(EMS)) and (h_i == heigth(EMS)) then
            remove EMS from the corresponding list
        else
            remove EMS from the corresponding list
            if (width(EMS) − w_i ≠ 0) then
                create EMS in the x-axis and create EMS in the EMS-front list
                length(EMS) = l_i
                width(EMS) = width(EMS) − w_i
                height(EMS) = h_i
            end if
            if (length(EMS) − l_i ≠ 0) then
                create EMS in the y-axis and create EMS in the EMS-beside list
                length(EMS) = length(EMS) − l_i
                width(EMS) = width(EMS)
                height(EMS) = h_i
            end if
            if (height(EMS) − h_i ≠ 0) then
                create EMS in the z-axis and create EMS in the EMS-above list
                length(EMS) = length(EMS)
                width(EMS) = width(EMS)
                height(EMS) = height(EMS) − h_i
            end if
        end if
        place the piece inside the container
        totalVolume = totalVolume + v_i
        totalWeight = totalWeight + p_i
        remove q_x from pieces
    end while
end while
```

Algorithm 9 Multiple-Level Filling Heuristic in Depth

pieces = $q_1, q_2, \ldots, q_N$
freeLength = W
totalVolume = 0
totalWeight = 0
$EMS_x = \emptyset$ EMS-beside list
$EMS_y = \emptyset$ EMS-front list
$EMS_z = \emptyset$ EMS-above list
while ($pieces \neq \emptyset$) and ($\exists q_x \subset pieces, q_x \in t_i, i \in [1,m]$ / ($totalWeight + p_i \leq P_{max}$) and ($totalVolume + v_i \leq (LxWxH)$) **do**
  while ($\exists q_x \subset pieces, q_x \in t_i, i \in [1,m]$ / $EMS \in EMS-beside$ or $EMS \in EMS-front$ or $EMS \in EMS-above$, $l_i \leq length(EMS)$, $w_i \leq width(EMS)$, $h_i \leq heigth(EMS)$) and ($totalWeight + p_i \leq P_{max}$) **do**
    if ($l_i == length(EMS)$) and ($w_i == widthh(EMS)$) and ($h_i == heigth(EMS)$) **then**
      remove EMS from the corresponding list
    else
      remove EMS from the corresponding list
      if ($width(EMS) - w_i \neq 0$) **then**
        create EMS in the x-axis and create EMS in the EMS-front list
        length(EMS) = l_i
        width(EMS) = $width(EMS) - w_i$
        height(EMS) = h_i
      end if
      if ($length(EMS) - l_i \neq 0$) **then**
        create EMS in the y-axis and create EMS in the EMS-beside list
        length(EMS) = $length(EMS) - l_i$
        width(EMS) = $width(EMS)$
        height(EMS) = h_i
      end if
      if ($height(EMS) - h_i \neq 0$) **then**
        create EMS in the z-axis and create EMS in the EMS-above list
        length(EMS) = $length(EMS)$
        width(EMS) = $width(EMS)$
        height(EMS) = $height(EMS) - h_i$
      end if
    end if
    place the piece inside the container
    $totalVolume = totalVolume + v_i$
    $totalWeight = totalWeight + p_i$
    remove q_x from pieces
  end while
  if ($pieces \neq \emptyset$) and ($\exists q_x \subset pieces$, $q_x \in t_i, i \in [1,m]$ / ($freeLength \geq l_i$) and ($q_x \nexists x_i, x_i \in [0, b_i], i \in [1,m]$)) **then**
    create a level (wall) with width(wall) = W, length(wall) = l_i, height(wall) = H
    freeLength = freeLength - l_i
    $q_x \in x_i$ // place the piece inside the container
    totalVolume = totalVolume + v_i
    totalWeight = totalWeight + p_i
    if (width(wall) - $w_i \neq 0$) **then**
      create EMS in the x-axis and create EMS in the EMS-front list
      length(wall) = l_i
      width(wall) = $W - w_i$
      height(wall) = h_i
    end if
    if (height(wall) - $h_i \neq 0$) **then**
      create EMS in the z-axis and create EMS in the EMS-above list
      length(wall) = l_i
      width(wall) = W
      height(wall) = $H - h_i$
    end if
    remove q_x from pieces
  end if
end while

Appendix

C

Problem Instances

Instance	Different type boxes	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} (Kg.)
$1C-01$	2	$[5-10]$	10	13718	$3.01e^{07}$	$3.31e^{07}$
$1C-02$	2	$[5-10]$	20	2000	$3.01e^{07}$	$3.61e^{07}$
$1C-03$	2	$[5-10]$	30	2662	$3.01e^{07}$	$3.91e^{07}$
$1C-04$	2	$[15-20]$	10	250	$3.01e^{07}$	$3.31e^{07}$
$1C-05$	2	$[15-20]$	20	250	$3.01e^{07}$	$3.61e^{07}$
$1C-06$	2	$[15-20]$	30	250	$3.01e^{07}$	$3.91e^{07}$
$1C-07$	2	$[25-30]$	10	54	$3.01e^{07}$	$3.31e^{07}$
$1C-08$	2	$[25-30]$	20	54	$3.01e^{07}$	$3.61e^{07}$
$1C-09$	2	$[25-30]$	30	54	$3.01e^{07}$	$3.91e^{07}$
$1C-10$	5	$[5-10]$	10	3124	$3.01e^{07}$	$3.31e^{07}$
$1C-11$	5	$[5-10]$	20	6894	$3.01e^{07}$	$3.61e^{07}$
$1C-12$	5	$[5-10]$	30	3359	$3.01e^{07}$	$3.91e^{07}$
$1C-13$	5	$[15-20]$	10	380	$3.01e^{07}$	$3.31e^{07}$
$1C-14$	5	$[15-20]$	20	848	$3.01e^{07}$	$3.61e^{07}$
$1C-15$	5	$[15-20]$	30	593	$3.01e^{07}$	$3.91e^{07}$
$1C-16$	5	$[25-30]$	10	180	$3.01e^{07}$	$3.31e^{07}$
$1C-17$	5	$[25-30]$	20	180	$3.01e^{07}$	$3.61e^{07}$
$1C-18$	5	$[25-30]$	30	180	$3.01e^{07}$	$3.91e^{07}$
$1C-19$	10	$[5-10]$	10	7599	$3.01e^{07}$	$3.31e^{07}$
$1C-20$	10	$[5-10]$	20	5946	$3.01e^{07}$	$3.61e^{07}$
$1C-21$	10	$[5-10]$	30	6219	$3.01e^{07}$	$3.91e^{07}$
$1C-22$	10	$[15-20]$	10	924	$3.01e^{07}$	$3.31e^{07}$
$1C-23$	10	$[15-20]$	20	2595	$3.01e^{07}$	$3.61e^{07}$
$1C-24$	10	$[15-20]$	30	1316	$3.01e^{07}$	$3.91e^{07}$
$1C-25$	10	$[25-30]$	10	216	$3.01e^{07}$	$3.31e^{07}$
$1C-26$	10	$[25-30]$	20	216	$3.01e^{07}$	$3.61e^{07}$
$1C-27$	10	$[25-30]$	30	216	$3.01e^{07}$	$3.91e^{07}$
$1C-28$	15	$[5-10]$	10	9656	$3.01e^{07}$	$3.31e^{07}$
$1C-29$	15	$[5-10]$	20	6156	$3.01e^{07}$	$3.61e^{07}$
$1C-30$	15	$[5-10]$	30	9775	$3.01e^{07}$	$3.91e^{07}$
$1C-31$	15	$[15-20]$	10	1108	$3.01e^{07}$	$3.31e^{07}$
$1C-32$	15	$[15-20]$	20	1185	$3.01e^{07}$	$3.61e^{07}$
$1C-33$	15	$[15-20]$	30	1047	$3.01e^{07}$	$3.91e^{07}$
$1C-34$	15	$[25-30]$	10	405	$3.01e^{07}$	$3.31e^{07}$
$1C-35$	15	$[25-30]$	20	405	$3.01e^{07}$	$3.61e^{07}$
$1C-36$	15	$[25-30]$	30	405	$3.01e^{07}$	$3.91e^{07}$

Table C.1: Problem instances: *1C*

Instance	Different type boxes	$[D_l - D_h]$	W_G	N	V_{max} (cm^3)	P_{max} $(Kg.)$
$1A-01$	2	$[5-10]$	10	2000	$6.15e^{07}$	$6.77e^{07}$
$1A-02$	2	$[5-10]$	20	3456	$6.15e^{07}$	$7.38e^{07}$
$1A-03$	2	$[5-10]$	30	4394	$6.15e^{07}$	$7.99e^{07}$
$1A-04$	2	$[15-20]$	10	432	$6.15e^{07}$	$6.77e^{07}$
$1A-05$	2	$[15-20]$	20	250	$6.15e^{07}$	$7.38e^{07}$
$1A-06$	2	$[15-20]$	30	250	$6.15e^{07}$	$7.99e^{07}$
$1A-07$	2	$[25-30]$	10	54	$6.15e^{07}$	$6.77e^{07}$
$1A-08$	2	$[25-30]$	20	54	$6.15e^{07}$	$7.38e^{07}$
$1A-09$	2	$[25-30]$	30	54	$6.15e^{07}$	$7.99e^{07}$
$1A-10$	5	$[5-10]$	10	5371	$6.15e^{07}$	$6.77e^{07}$
$1A-11$	5	$[5-10]$	20	2731	$6.15e^{07}$	$7.38e^{07}$
$1A-12$	5	$[5-10]$	30	5079	$6.15e^{07}$	$7.99e^{07}$
$1A-13$	5	$[15-20]$	10	380	$6.15e^{07}$	$6.77e^{07}$
$1A-14$	5	$[15-20]$	20	413	$6.15e^{07}$	$7.38e^{07}$
$1A-15$	5	$[15-20]$	30	750	$6.15e^{07}$	$7.99e^{07}$
$1A-16$	5	$[25-30]$	10	180	$6.15e^{07}$	$6.77e^{07}$
$1A-17$	5	$[25-30]$	20	180	$6.15e^{07}$	$7.38e^{07}$
$1A-18$	5	$[25-30]$	30	180	$6.15e^{07}$	$7.99e^{07}$
$1A-19$	10	$[5-10]$	10	5397	$6.15e^{07}$	$6.77e^{07}$
$1A-20$	10	$[5-10]$	20	7569	$6.15e^{07}$	$7.38e^{07}$
$1A-21$	10	$[5-10]$	30	4313	$6.15e^{07}$	$7.99e^{07}$
$1A-22$	10	$[15-20]$	10	1184	$6.15e^{07}$	$6.77e^{07}$
$1A-23$	10	$[15-20]$	20	885	$6.15e^{07}$	$7.38e^{07}$
$1A-24$	10	$[15-20]$	30	717	$6.15e^{07}$	$7.99e^{07}$
$1A-25$	10	$[25-30]$	10	216	$6.15e^{07}$	$6.77e^{07}$
$1A-26$	10	$[25-30]$	20	216	$6.15e^{07}$	$7.38e^{07}$
$1A-27$	10	$[25-30]$	30	216	$6.15e^{07}$	$7.99e^{07}$
$1A-28$	15	$[5-10]$	10	7533	$6.15e^{07}$	$6.77e^{07}$
$1A-29$	15	$[5-10]$	20	9416	$6.15e^{07}$	$7.38e^{07}$
$1A-30$	15	$[5-10]$	30	5473	$6.15e^{07}$	$7.99e^{07}$
$1A-31$	15	$[15-20]$	10	781	$6.15e^{07}$	$6.77e^{07}$
$1A-32$	15	$[15-20]$	20	837	$6.15e^{07}$	$7.38e^{07}$
$1A-33$	15	$[15-20]$	30	762	$6.15e^{07}$	$7.99e^{07}$
$1A-34$	15	$[25-30]$	10	405	$6.15e^{07}$	$6.77e^{07}$
$1A-35$	15	$[25-30]$	20	405	$6.15e^{07}$	$7.38e^{07}$
$1A-36$	15	$[25-30]$	30	405	$6.15e^{07}$	$7.99e^{07}$

Table C.2: Problem instances: *1A*

Bibliography

[1] 3D LoadPacker. http://astrokette.com. (Visited on 26/03/2017).

[2] AutoLoad Pro. http://www.autoload-pro-container-loading-software.soft112.com. (Visited on 26/03/2017).

[3] Cargomanager. http://www.packyourcontainer.com. (Visited on 26/03/2017).

[4] CargoWiz. http://www.softtruck.com. (Visited on 26/03/2017).

[5] Cube-iq. http://www.magiclogic.com. (Visited on 26/03/2017).

[6] Cubemaster. http://www.logensol.com. (Visited on 26/03/2017).

[7] Loadmaster. http://zhuangxiang.com/en/features.html. (Visited on 26/03/2017).

[8] Loadplanner. http://loadplanner.com. (Visited on 26/03/2017).

[9] Maxload pro. http://www.topseng.com/. (Visited on 26/03/2017).

[10] Packvol. http://packvol.com. (Visited on 26/03/2017).

[11] Unity 3d. http://unity3d.com. (Visited on 26/03/2017).

[12] El contenedor y su tipología. *Puertos del Estado. Boletín información mensual N.54. Almazan Ingenieros*, Junio 1998.

[13] F. A. Agrama. Multi-objective genetic optimization of linear construction projects. *HBRC Journal*, 8(2):144 – 151, 2012.

[14] G. Aguado Sarrió. Posibilidades de desarrollo de videojuegos fps utilizando tecnología webgl, 2015/2016.

[15] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, C. León, G. Luque, J. Petit, C. Rodríguez, A. Rojas, and F. Xhafa. Efficient parallel LAN/WAN algorithms for optimization. The MALLBA project. *Parallel Computing*, 32(5-6):415–440, June 2006.

[16] E. Alba, G. Luque, and S. Nesmachnow. Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2013.

[17] A. F. Ali and A.-E. Hassanien. *A Survey of Metaheuristics Methods for Bioinformatics Applications*, pages 23–46. Springer International Publishing, Cham, 2016.

[18] S. Allen, E. Burke, and G. Kendall. A hybrid placement strategy for the three-dimensional strip packing problem. *European Journal of Operational Research*, 209(3):219–227, 2011.

[19] A. d. Almeida and M. B. Figueiredo. A particular approach for the Three-dimensional Packing Problem with additional constraints. *Computers and OR*, 37:1968–1976, November 2010.

[20] M. T. Alonso, R. Alvarez-Valdes, J. M. Tamarit, and F. Parreño. A reactive grasp algorithm for the container loading problem with load-bearing constraints. *European Journal of Industrial Engineering*, 8(5):669–694, 2014.

[21] H. Alvarez Campos. Mediación instrumental basada en ar (realidad aumentad), sobre modelado de buques - competencias para la formación tecnológica. In *XIV Encuentro internacional Virtual Educa Colombia*, June 2013.

[22] R. R. Amossen and D. Pisinger. Multi-dimensional bin packing problems with guillotine constraints. *Computers & Operations Research*, 37(11):1999 – 2006, 2010. Metaheuristics for Logistics and Vehicle Routing.

[23] A. E. Ansary and M. Shalaby. Evolutionary multi-objective optimization for positioning of residential houses. *International Journal of Civil, Environmental, Structural, Construction and Architectural Engineering*, 6(7):124 – 130, 2012.

[24] O. C. Araújo and V. A. Armentano. A multi-start random constructive heuristic for the container loading problem. *Pesquisa Operacional*, 27(2), Aug. 2007.

[25] S. Balakirsky, F. Proctor, T. Kramer, P. Kolhe, and H. I. Christensen. *Using Simulation to Assess the Effectiveness of Pallet Stacking Methods*, pages 336–349. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[26] S. Bechikh, R. Datta, and A. Gupta. *Recent Advances in Evolutionary Multi-objective Optimization*. Adaptation, Learning, and Optimization. Springer International Publishing, 2016.

[27] D. Bernhofen, Z. El-Sahli, and R. Kneller. Estimating the effects of the container revolution on world trade. CESifo Working Paper Series 4136, CESifo Group Munich, 2013.

[28] E. Bischoff. Three-dimensional packing of items with limited load bearing strength. *European Journal of Operational Research*, 168(3):952 – 966, 2006. Balancing Assembly and Transfer lines.

[29] E. Bischoff and M. Ratcliff. Issues in the development of approaches to container loading. *Omega*, 23(4):377 – 390, 1995.

[30] E. Bischoff and M. Ratcliff. Loading Multiple Pallets. volume 46, pages 1322–1336, Nov. 1995.

[31] E. E. Bischoff, F. Janetz, and M. S. W. Ratcliff. Loading pallets with non-identical items. *European Journal of Operational Research*, 84(3):681–692, August 1995.

[32] E. E. Bischoff and M. D. Marriott. A comparative evaluation of heuristics for container loading. *European Journal of Operational Research*, 44(2):267 – 276, 1990.

[33] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA - A Platform and Programming Language Independent Interface for Search Algorithms. In *Conference on Evolutionary Multi-Criterion Optimization*, volume 2632 of *LNCS*, pages 494–508, Faro, Portugal, 2003. Springer.

[34] C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[35] A. Bortfeldt and H. Gehring. A tabu search algorithm for weakly heterogeneous container loading problems. *OR Spectrum*, 20(4):237–250, December 1998.

[36] A. Bortfeldt and H. Gehring. *Applying Tabu Search to Container Loading Problems*, pages 533–538. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[37] A. Bortfeldt and H. Gehring. Two metaheuristics for strip packing problems. In *Proceedings band der 5th International Conference of the Decision Sciences Institute*, volume 2, pages 1153–1156, Athen, 1999.

[38] A. Bortfeldt and H. Gehring. A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, 131(1):143–161, May 2001.

[39] A. Bortfeldt and H. Gehring. New Large Benchmark Instances for the Two-Dimensional Strip Packing Problem with Rectangular Pieces. In *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 30.2. IEEE Computer Society, 2006.

[40] A. Bortfeldt, H. Gehring, and D. Mack. A parallel tabu search algorithm for solving the container problem. *Parallel Computing*, 29:641–662, 2003.

[41] A. Bortfeldt and S. Jungmann. A tree search algorithm for solving the multi-dimensional strip packing problem with guillotine cutting constraint. *Annals of Operations Research*, 196(1):53–71, 2012.

[42] A. Bortfeldt and D. Mack. A heuristic for the three-dimensional strip packing problem. *European Journal of Operational Research*, 183(3):1267–1279, 2007.

[43] A. Bortfeldt and G. Wäscher. Container Loading Problems - A State-of-the-Art Review. Working Paper 1, Otto-von-Guericke-Universität Magdeburg, April 2012.

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

[45] I. BoussaïD, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Inf. Sci.*, 237:82–117, July 2013.

[46] R. Brooks, C. Smith, A. Stone, and W. Tutte. The dissection of rectangles into squares. *Duke Mathematical Journal*, 7:312–340, 1940.

[47] M. Brusco, G. Thompson, and L. Jacobs. A morph-based simulated annealing heuristic for a modified bin-packing problem. *Journal of the Operational Research Society*, 48(4):433–439, April 1997.

[48] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles*, 10, 1998.

[49] R. Carraway, T. Morin, and H.Moskowitz. Generalized dynamic programming for multicriteria optimization. pages 95–104. European Journal of Operational Research, 1990.

[50] S. Ceschia and A. Schaerf. Local search for a multi-drop multi-container loading problem. *Journal of Heuristics*, 19(2):275–294, 2013.

[51] F. T. Chan, R. Bhagwat, N. Kumar, M. Tiwari, and P. Lam. Development of a decision support system for air-cargo pallets loading problem: A case study. *Expert Systems with Applications*, 31(3):472 – 485, 2006.

[52] C. Chen, S. Lee, and Q. Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80(1):68 – 76, 1995.

[53] D. Chen, J. Liu, Y. Fu, and M. Shang. An efficient heuristic algorithm for arbitrary shaped rectilinear block packing problem. *Comput. Oper. Res.*, 37:1068–1074, June 2010.

[54] S. G. Christensen and D. M. Rousøe. Container loading with multi-drop constraints. *International Transactions in Operational Research*, 16(6):727–743, 2009.

[55] C. A. 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*, 191(11-12):1245–1287, January 2002.

[56] C. A. Coello and G. B. Lamont, editors. *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific, Singapore, 2004.

[57] C. A. Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic and Evolutionary Computation. Springer, 2007.

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

[59] C. Darwin. The origin of species by means of natural selection or the preservation of favoured races in thestruggle for life. Cambridge University Press, 1964.

[60] A. Davies and E. E. Bischoff. Weight distribution considerations in container loading. *European Journal of Operational Research*, 114(3):509 – 527, 1999.

[61] J. de Armas, G. Miranda, and C. León. Two encoding schemes for a multi-objective cutting stock problem. In *2011 IEEE Congress on Evolutionary Computation (CEC)*, pages 27–109, New Orleans, LA, June 2011.

[62] J. de Armas, G. Miranda, C. Leon, and C. Segura. Optimisation of a Multi-Objective Two-Dimensional Strip Packing Problem based on Evolutionary Algorithms. *International Journal of Production Research*, 48(7):2011– 2028, 2009.

[63] De Castro Silva J. L. and Soma, N. Y. and Maculan, N. A greedy search for the three-dimensional bin packing problem: the packing static stability case. *International Transactions in Operational Research*, 10(2):141–153, 2003.

[64] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, Chichester, U.K., 2001.

[65] K. Deb and R. B. Agrawal. Simulated Binary Crossover for Continuous Search Space. *Complex Systems*, 9:115–148, 1995.

[66] K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[67] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: Nsga-ii. *Trans. Evol. Comp*, 6(2):182–197, Apr. 2002.

[68] DEI - Operations Research Group. Library of Instances: Bin Packing Problem. (Visited on 20/02/2017).

[69] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[70] T. Dereli and G. Sena Das. A Hybrid Simulated Annealing Algorithm for Solving Multi-objective Container Loading Problems. *Applied Artificial Intelligence: An International Journal*, 24(5):463–486, 2010.

[71] T. Dereli and G. Sena Das. A hybrid bee(s) algorithm for solving container loading problems. *Applied Soft Computing*, 11(2):2854–2862, 2011.

[72] X. Ding, Y. Han, and X. Zhang. A discussion of adaptive genetic algorithm solving container-loading problem. *Periodical of Ocean University of China*, 34(5):844–848.

[73] K. A. Dowsland and W. B. Dowsland. Packing Problems. *European Journal of Operational Research*, 56(1):2–14, 1992.

[74] J. J. Durillo, A. J. Nebro, F. Luna, B. Dorronsoro, and E. Alba. jMetal: a Java Framework for Developing Multi-Objective Optimization Metaheuristics. Technical Report ITI-2006-10, Universidad de Málaga, 2006.

[75] H. Dyckhoff. A Typology of Cutting and Packing Problems. *European Journal of Operational Research*, 44(2):145–159, 1990.

[76] H. Dyckhoff and U. Finke. *Cutting and Packing in Production and Distribution: A Typology and Bibliography*. Springer, 1992.

[77] H. Dyckhoff, G. Scheithauer, and J. Terno. *Annotated bibliographies in combinatorial optimization*, chapter Cutting and packing, pages 393–414. Wiley, New York, 1997.

[78] J. Egeblad, C. Garavelli, S. Lisi, and D. Pisinger. Heuristics for container loading of furniture. *European Journal of Operational Research*, 200(3):881–892, 2010.

[79] J. Egeblad, B. K. Nielsen, and M. Brazil. Translational packing of arbitrary polytopes. *Computational Geometry*, 42(4):269 – 288, 2009.

[80] J. Egeblad, B. K. Nielsen, and A. Odgaard. Fast neighborhood search for two- and three-dimensional nesting problems. *European Journal of Operational Research*, 183(3):1249 – 1266, 2007.

[81] M. Ehrgott. Vilfredo Pareto and Multi-objective Optimization. *Documenta Mathematica*, Extra Volu(1):447–453, 2012.

[82] A. E. Eiben. *Handbook of Evolutionary Computation.* IOP Publishing Ltd. and Oxford University Press, 1998.

[83] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing.* Natural computing series. Springer, Berlin, Heidelberg, Paris, 2003. Réimpression 2007.

[84] M. Eley. Solving container loading problems by block arrangement. *European Journal of Operational Research*, 141(2):393–409, 2002.

[85] M. Eley. A bottleneck assignment approach to the multiple container loading problem. *OR Spectrum*, 25(1):45–60, 2003.

[86] M. Emmerich and R. Hosenberg. TEA - A Toolbox for the Design of Parallel Evolutionary Algorithms in C++. Technical Report CI-106/01, SFB 531, University of Dortmund, Germany, 2001.

[87] H. A. Erdem. Solving container loading problem with genetic algorithm. In *2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 391–396, Nov 2014.

[88] D. Erni and et al. Application of evolutionary optimization algorithms in computational optics, 2000.

[89] EURO Special Interest Group on Cutting and Packing. http://paginas.fe.up.pt/∼esicup. (Visited on 20/02/2017).

[90] L. Faina. A global optimization algorithm for the three-dimensional packing problem. *European Journal of Operational Research*, 126(2):340–354, October 2000.

[91] T. Fanslau and A. Bortfeldt. A Tree Search Algorithm for Solving the Container Loading Problem. *Informs Journal on Computing*, 22(2):222–235, 2010.

[92] G. Fasano. *Cargo Analytical Integration in Space Engineering: A Three-dimensional Packing Model*, pages 232–246. Palgrave Macmillan UK, London, 1999.

[93] G. Fasano. *MIP Models for Solving 3-Dimensional Packing Problems Arising in Space Engineering*, pages 43–56. Springer US, Boston, MA, 2003.

[94] J. M. Fernández and B. Barén. Equipo de Algoritmos Evolutivos Multiobjetivo Paralelos. Universidad Nacional de Asunción. Asunción, Paraguay, 2005.

[95] J. M. Fernández and B. Barén. Equipo Elitista de Algoritmos Evolutivos Multiobjetivo. In *Proceedings of CLEI'2005. Latin-American Conference on Informatics*. Cali, Colombia, ISSN 0717-500, 2005.

[96] J. M. Fernández and B. Barèn. Equipo Probabilìstico de Algoritmos Evolutivos Multiobjetivo en Computaciòn Paralela. In *Symposium on Artificial Intelligence, ASAI 2005*. Rosario, Argentina, 2005.

[97] L. Flavell. *Beginning Blender: Open Source 3D Modeling, Animation, and Game Design*. Apresspod Series. Apress, 2010.

[98] C. Fonseca and P. J. Fleming. On the performance assessment and comparison of stochastic multiobjective optimizers. In *Parallel Problem Solving from Nature*, LNCS, pages 584–593. Springer-Verlag, 1996.

[99] C. M. Fonseca and P. J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

[100] G. Fuellerer, K. F. Doerner, R. F. Hartl, and M. Iori. Metaheuristics for vehicle routing problems with three-dimensional loading constraints. *European Journal of Operational Research*, 201(3):751 – 759, 2010.

[101] C. Gagné and M. Parizeau. Genericity in Evolutionary Computation Software Tools: Principles and Case Study. *International Journal on Artificial Intelligence Tools*, 15(2):173–194, 2006.

[102] J. F. C. García. Anteproyecto. incorporación en matlab de algoritmos metaheurísticos.

[103] L. D. Gaspero and A. Schaerf. EasyLocal++: an object-oriented framework for the flexible design of local search algorithms and metaheuristics. In *4th Metaheuristics International Conference*, pages 287–292, 2001.

[104] Z. Gee and P. Falco. *3D in Photoshop: The Ultimate Guide for Creative Professionals*. Focal Press/Elsevier, 2010.

[105] H. Gehring and A. Bortfeldt. A Genetic Algorithm for Solving the Container Loading Problem. *International Transactions in Operational Research*, 4(5-6):401–418, 1997.

[106] H. Gehring and A. Bortfeldt. A Parallel Genetic Algorithm for Solving the Container Loading Problem. *International Transactions in Operational Research*, 9(4):497–511, 2002.

[107] H. Gehring, K. Menschner, and M. Meyer. A computer-based heuristic for packing pooled shipment containers. *European Journal of Operational Research*, 44(2):277–288, 1990.

[108] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40(3):342–350, Aug. 2006.

[109] J. A. George and D. Robinson. A heuristic for packing boxes into a container. *Computers and Operations Research*, 7(3):147–156, 1980.

[110] I. Giagkiozis and P. Fleming. Methods for multi-objective optimization: An analysis. *Information Sciences*, 293:338 – 350, 2015.

[111] P. C. Gilmore and R. E. Gomory. Multistage Cutting Stock Problems of Two and More Dimensions. *OPERATIONS RESEARCH*, 13(1):94–120, Jan. 1965.

[112] P. C. Gilmore and R. E. Gomory. The theory and computation of knapsack functions. *Operations Research*, 14(6):1045–1074, 1966.

[113] J. F. Gonçalves and M. G. C. Resende. A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization*, 22(2):180–201, 2011.

[114] J. F. Gonçalves and M. G. Resende. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Comput. Oper. Res.*, 39(2):179–190, Feb. 2012.

[115] J. J. M. Guervós. Algoritmos evolutivos en perl. http://es.tldp.org/Presentaciones/200211hispalinux/merelo/ae-hispalinux2002.pdf.

[116] R. W. Haessler and F. Brian Talbot. Load planning for shipments of low density products. *European Journal of Operational Research*, 44(2):289–299, 1990.

[117] D. He, J. Cha, and Y. Jiang. Research on solution to complex container loading problem based on genetic algorithm. *Journal of Software*, 12(9):1380–1385, 2001.

[118] K. He and W. Huang. An efficient placement heuristic for three-dimensional rectangular packing. *Comput. Oper. Res.*, 38:227–233, January 2011.

[119] J. Hemminki and U. of Turku. Institute for Applied Mathematics. *Container Loading with Variable Strategies in Each Layer*. University of Turku, 1994.

[120] M. Hifi. Exact algorithms for unconstrained three-dimensional cutting problems: a comparative study. *Comput. Oper. Res.*, 31:657–674, April 2004.

[121] A. I. Hinxman. The trim-loss and assortment problems: A survey. *European Journal of Operational Research*, 5:8–18, 1979.

[122] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[123] H. Horii, M. Miki, T. Koizumi, and N. Tsujiuchi. Asynchronous Migration of Island Parallel GA for Multi-Objective Optimization Problem. In *Asia-Pacific Conference on Simulated Evolution and Learning*, volume 1, pages 86–90, 2002.

[124] W. Huang and K. He. A caving degree approach for the single container loading problem. *European Journal of Operational Research*, 196(1):93–101, 2009.

[125] W. Huang and K. He. A new heuristic algorithm for cuboids packing with no orientation constraints. *Computers & Operations Research*, 36(2):425 – 432, 2009. Scheduling for Modern Manufacturing, Logistics, and Supply Chains.

[126] D. Hummels. Transportation Costs and International Trade in the Second Era of Globalization. *Journal of Economic Perspectives*, 21(3):131–154, 2007.

[127] M. Iori and S. Martello. Routing problems with loading constraints. *TOP*, 18(1):4–27, 2010.

[128] N. Ivancic, K. Mathur, and B. B. Mohanty. An Integer Programming Based Heuristic Approach to the Three-dimensional Packing Problem. 1989.

[129] Z. Jin, K. Ohno, and J. Du. An efficient approach for the three dimensional container packing problem with practical constraints. *Asia-Pacific Journal of Operational Research*, 21(3):279–295, 2004.

[130] L. Junqueira, R. Morabito, and D. Sato Yamashita. Mip-based approaches for the container loading problem with multi-drop constraints. *Annals of Operations Research*, 199(1):51–75, 2012.

[131] L. Junqueira, R. Morabito, and D. S. Yamashita. Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers & Operations Research*, 39(1):74 – 85, 2012. Special Issue on Knapsack Problems and Applications.

[132] A. Kafafy. *Hybrid Evolutionary Metaheuristics for Multiobjective Decision Support*. Theses, Université Claude Bernard - Lyon I, Oct. 2013.

[133] L. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6:366–422, 1960 (Russian version appeared in 1939).

[134] J. Karelahti. Solving the cutting stock problem in the steel industry. Master's thesis, Department of Engineering Physics and Mathematics, Helsinki University of Technology, Helsinki, Finland, 2002.

[135] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer. Evolving objects: A general purpose evolutionary computation library. In *Artificial Evolution*, pages 231–244, 2001.

[136] J. Knowles. A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjetive optimizers. In *Intelligent System Design and Applications, 2005. ISDA'05. Proceedings. 5th International Conference on*, pages 552–557, 2005.

[137] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evol. Comput.*, 8(2):149–172, June 2000.

[138] I. Kroo. Aeronautical applications of evolutionary design.

[139] K. K. Lai and J. W. M. Chan. Developing a simulated annealing algorithm for the cutting stock problem. *Comput. Ind. Eng.*, 32(1):115–127, Jan. 1997.

[140] C. León, G. Miranda, and C. Rodríguez. *Optimization techniques for solving complex problems*, chapter Tools for tree searches: Branch-and-Bound and A* Algorithms, pages 193–208. Wiley, 2008.

[141] C. León, G. Miranda, and C. Segura. Parallel Hyperheuristic: A Self-Adaptive Island-Based Model for Multi-Objective Optimization. In *Genetic and Evolutionary Computation Conference*, pages 757–758, Atlanta, U.S.A., July 2008. ACM.

[142] C. León, G. Miranda, and C. Segura. METCO: A Parallel Plugin-Based Framework for Multi-Objective Optimization. *International Journal on Artificial Intelligence Tools*, 18(4):569–588, 2009.

[143] C. León, G. Miranda, E. Segredo, and C. Segura. Librería Paralela de Algoritmos Evolutivos Multi-Objetivo. In *XIX Jornadas de Paralelismo*, pages 3–8, Castellón, Spain, September 2008. Universitat Jaume I.

[144] C. León, G. Miranda, E. Segredo, and C. Segura. Equipo de Algoritmos Evolutivos con Mecanismo Auto-adaptativo para Optimización Multi-Objetivo. In E. Alba, F. Chicano, F. Luna, and G. Luque, editors, *VI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, pages 555–562, Málaga, Spain, February 2009. Universidad de Málaga.

[145] C. León, G. Miranda, E. Segredo, and C. Segura. Parallel Library of Multi-objective Evolutionary Algorithms. In D. E. Baz, F. Spies, and T. Gross, editors, *17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 28–35, Weimar, Germany, February 2009. IEEE Computer Society.

[146] A. Liefooghe, M. Basseur, L. Jourdan, and E.-G. Talbi. ParadisEO-MOEO: A Framework for Evolutionary Multi-objective Optimization. In *Evolutionary Multi-Criterion Optimization*, volume 4403 of *LNCS*, pages 386–400. Springer, 2007.

[147] A. Lim, B. Rodrigues, and Y. Yang. 3-D Container Packing Heuristics. *Applied Intelligence*, 22:125–134, March 2005.

[148] J. Liu, Z. Dong, and G. Ma. Solving container loading based on tabu search algorithm. *Journal of Shenyang University of Technology*, 31(2):212–216, 2009.

[149] J. Liu, Y. Yue, Z. Dong, C. Maple, and M. Keech. A novel hybrid tabu search approach to container loading. *Comput. Oper. Res.*, 38:797–807, April 2011.

[150] S. Liu, W. Tan, Z. Xu, and X. Liu. A tree search algorithm for the container loading problem. *Computers & Industrial Engineering*, 75:20–30, 2014.

[151] A. Lodi, S. Martello, and M. Monaci. Two-Dimensional Packing Problems: A Survey. *European Journal of Operational Research*, 141(2):241–252, September 2002.

[152] T. H. Loh and A. Y. C. Nee. A packing algorithm for hexahedral boxes. pages 115–126, 1992.

[153] D. Mack, A. Bortfeldt, and H. Gehring. A parallel hybrid local search algorithm for the container loading problem. *International Transactions in Operational Research*, 11(5):511–533, 2004.

[154] O. C. Makarem and R. A. Haraty. Smart container loading. *J. Comput. Meth. in Science and Engineering*, 10(3-6):231–245, 2010.

[155] M. Márquez Gómez. Las metaheurísticas: tendencias actualesny su aplicabilidad en la ergonoía. volume IV, pages 108–120, 2013.

[156] R. Mart and G. Reinelt. *The Linear Ordering Problem: Exact and Heuristic Methods in Combinatorial Optimization*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[157] S. Martello, D. Pisinger, and D. Vigo. The Three-Dimensional Bin Packing Problem. *Operations Research*, 48(2):256–267, 2000.

[158] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons Ltd, 1990.

[159] D. A. Martínez, R. Alvarez-Valdes, and F. Parreño. A GRASP Algorithm for the Container Loading Problem with Multi-Drop Constraints. *Pesquisa Operacional*, 35:1 − 24, 04 2015.

[160] G. Miranda. *Algorithmic Skeletons for Combinatorial Optimization. Parallelizations and Applications*. PhD thesis, Universidad de La Laguna, La Laguna, Spain, May 2009.

[161] G. Miranda, J. de Armas, C. Segura, and C. León. Hyperheuristic codification for the multi-objective 2d guillotine strip packing problem. In *IEEE Congress on Evolutionary Computation*, pages 1–8, July 2010.

[162] U. Mishra. *Improving Graphical User Interface Using TRIZ:*. Umakanta Mishra, 2009.

[163] R. Morabito and M. Arenalest. An AND/OR-graph Approach to the Container Loading Problem. *International Transactions in Operational Research*, 1(1):59–73, 1994.

[164] A. Moura and J. F. Oliveira. A grasp approach to the container-loading problem. *IEEE Intelligent Systems*, 20(4):50–57, July 2005.

[165] V. Neidlein and G. Wäscher. SLOPPGEN: A Problem Generator for the Two-Dimensional Rectangular Single Large Object Placement Problem With a Single Defect. FEMM Working Papers 08015, Otto-von-Guericke University Magdeburg, Faculty of Economics and Management, July 2008.

[166] B. Ngoi, M. Tay, and E. Chua. Applying spatial representation techniques to the container packing problem. *International Journal of Production Research*, 32(1):111–123, 1994.

[167] K. Nirmal and N. Mishra. 3D WIMP put into action for 3D GUI. *International Journal of Emerging Technology and Advanced Engineering*, 3(101):598–605, April 2013.

[168] K. R. Nirmal and N. Mishra. 3d graphical user interface on personal computer using p5 data glove. *International Journal of Computer Science Issues*, 8(5), 2011.

[169] L. S. d. Oliveira and S. A. F. P. Saramago. Multiobjective optimization techniques applied to engineering problems. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 32:94 – 105, 03 2010.

[170] OR-Library. http://people.brunel.ac.uk/∼mastjjb/jeb/info.html. (Visited on 20/02/2017).

[171] PackLib². http://ibr.cs.tu-bs.de/alg/packlib. (Visited on 20/02/2017).

[172] F. Parreño, R. Alvarez-Valdes, J. Oliveira, and J. Tamarit. An hybrid grasp/vnd algorithm for two- and three-dimensional bin packing. *Annals of Operations Research*, 179:203–220, 2010.

[173] F. Parreño, R. Alvarez-Valdes, J. F. Oliveira, and J. M. Tamarit. Neighborhood structures for the container loading problem: a vns implementation. *Journal of Heuristics*, 16:1–22, February 2010.

[174] F. Parreòo, R. Alvarez-Valdes, J. M. Tamarit, and J. F. Oliveira. A maximal-space algorithm for the container loading problem. *Informs J. on Computing*, 20(3):412–422, July 2008.

[175] F. Parreño, R. Alvarez-Valdés, J. F. Oliveira, and J. M. Tamarit. Neighborhood structures for the container loading problem: a vns implementation. *J. Heuristics*, 16(1):1–22, 2010.

[176] J. E. Pérez. Introducción a javascript. http://www.librosweb.es/javascript. (Visited on 26/03/2017).

[177] A. Petrovski and J. McCall. *Multi-objective Optimisation of Cancer Chemotherapy Using Evolutionary Algorithms*, pages 531–545. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[178] D. Pisinger. Heuristics for the container loading problem. *European Journal of Operational Research*, 141(2):382–392, 2002.

[179] A. Ponsich, A. L. Jaimes, and C. A. C. Coello. A survey on multiobjective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications. *IEEE Transactions on Evolutionary Computation*, 17(3):321–344, June 2013.

[180] J. Puchinger and G. R. Raidl. *Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification*, pages 41–53. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[181] J. Ren, Y. Tian, and T. Sawaragi. A tree search method for the container loading problem with shipment priority. *European Journal of Operational Research*, 214(3):526 – 535, 2011.

[182] E. L. Ruiz, C. E. Izquierdo, B. M. Batista, and J. M. Moreno-Vega. *A Metaheuristic Approach for the Seaside Operations in Maritime Container Terminals*, pages 28–35. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[183] R. Sagarra, A. de Souza, J. Mollofre, and J. de Larrucea. *El transporte de contenedores: terminales, operatividad y casuística*. e-politext. Universitat Politecnica de Catalunya. Iniciativa Digital Politecnica, 2004.

[184] H. M. Salkin and C. A. de Kluyver. The Knapsack Problem: A survey*. *Naval Research Logistics Quartely*, (22):127–144, 1975.

[185] Y. Sawaragi, H. Nakayama, and T. Tanino. *Theory of multiobjective optimization*. Academic Press, Orlando, 1985.

[186] J. Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, 1984.

[187] J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette, editor, *International Conference on Genetic Algorithms and Their Applications*, pages 93–100, 1985.

[188] G. Scheithauer. Algorithms for the Container Loading Problem. In *Operations Research Proceedings 1991*, pages 445–452. Springer-Verlag, 1992.

[189] J. Schrïter. Graphic generation of setting patterns for robotic setting plants. *CFI Ceramic Forum International*, 83(1):E12–E13, 2006.

[190] A. Sciomachen and E. Tanfani. A 3d-bpp approach for optimising stowage plans and terminal productivity. *European Journal of Operational Research*, 183(3):1433 – 1446, 2007.

[191] E. Segredo. *Parameter Setting in Sequential and Parallel Evolutionary Algorithms. An Approach based on Hyper-heuristic and Parameter Control Strategies.* PhD thesis, Universidad de La Laguna, La Laguna, Spain, June 2014.

[192] C. Segura, A. Cervantes, A. J. Nebro, M. D. Jaraíz-Simón, E. Segredo, S. García, F. Luna, J. A. Gómez-Pulido, G. Miranda, C. Luque, E. Alba, M. A. Vega-Rodríguez, C. León, and I. Galván. Optimizing the DFCN Broadcast Protocol with a Parallel Cooperative Strategy of Multi-Objective Evolutionary Algorithms. In Springer-Verlag, editor, *5th International Conference devoted to Evolutionary Multi-Criterion Optimization*, volume 5467, pages 305–319, Nantes, France, April 2009.

[193] C. Segura, Y. González, G. Miranda, and C. León. *A Multi-Objective Evolutionary Approach for the Antenna Positioning Problem*, pages 51–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[194] T. Sen, M. Raiszadeh, and P. Dileepan. A branch and bound approach to the bicriterion scheduling problem involving total flowtime and range of lateness. Number 34, pages 254–260. Management Science, 1988.

[195] D. Sheskin. *The handbook of parametric and nonparametric statistical procedures.* CRC Press, 2003.

[196] E. Silva, J. F. Oliveira, and G. Wäscher. 2dcpackgen: A problem generator for two-dimensional rectangular cutting and packing problems. *European Journal of Operational Research*, 237(3):846 – 856, 2014.

[197] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman & Hall, London, 1986.

[198] S.keerthi, A. K, and V. M.v. Article: Survey paper on swarm intelligence. *International Journal of Computer Applications*, 115(5):8–12, April 2015. Full text available.

[199] S.-M. Soak, S.-W. Lee, and M. Jeon. The improved adaptive link adjustment evolutionary algorithm for the multiple container packing problem. *Applied Intelligence*, 33:144–158, October 2010.

[200] K. Sörensen, M. Sevaux, and F. Glover. A history of metaheuristics. In *ORBEL29 - 29th Belgian conference on Operations Research*, Proceedings of the ORBEL29 conference, Antwerp, Belgium, Feb. 2015. University of Antwerp - ANT/OR research group.

[201] N. Srinivas and K. Deb. Multiobjective Function Optimization using Nondominated sorting Genetic Algorithms. *Evolutionary Computation Journal*, 2(3):221–248, 1994.

[202] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. John Wiley, New York, 1986.

[203] B. Stewart and C. white. Multiobjective a*. Number 38, pages 775–814. Journal of the ACM, 1991.

[204] P. E. Sweeney and E. R. Paternoster. Cutting and Packing Problems: A categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.

[205] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[206] C. D. Tarantilis, E. E. Zachariadis, and C. T. Kiranoudis. A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. *IEEE Transactions on Intelligent Transportation Systems*, 10(2):255–271, June 2009.

[207] J. Terno, G. Scheithauer, U. Sommerweiß, and J. Riehme. An efficient approach for the multi-pallet loading problem. *European Journal of Operational Research*, 123:372–381, 1997.

[208] S. Tschöke and T. Polzer. Portable Parallel Branch-and-Bound Library - PPBB-LIB, 1996. http://wwwcs.uni-paderborn.de/fachbereich/AG/monien/SOFTWARE/PPBB/ppbblib.html.

[209] D. A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Wright Patterson AFB, OH, USA, 1999. AAI9928483.

[210] M. Vega, J. Marcos, M. B. Melián Batista, and J. A. Moreno Pérez. Metaheurísticas: Una visión global. *Inteligencia artificial: Revista Iberoamericana de Inteligencia Artificial*, 7(19):7–28, 2003.

[211] D. A. V. Veldhuizen, J. B. Zydallis, and G. B. Lamont. Considerations in engineering parallel multiobjective evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, 7(2):144–173, 2003.

[212] D. V. Veldhuizen. Multiobjective evolutionary algorithms: Classifications, analyzes, and new innovations. *PhD tesis, Graduate School of Eng. of the Air Force Inst. of Technol*, page 249, 1999.

[213] P. Y. Wang and C. L. Valenzuela. Data set generation for rectangular placement problems. *European Journal of Operational Research*, 134(2):378–391, October 2001.

[214] Z. Wang, K. W. Li, and J. K. Levy. A heuristic for the container loading problem: A tertiary-tree-based dynamic space decomposition approach. *European Journal of Operational Research*, 191(1):86–99, November 2008.

[215] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, December 2007.

[216] T. Weise. *Global Optimization Algorithms - Theory and Application*. 2008.

[217] H. Wu, S. C. Leung, Y. whar Si, D. Zhang, and A. Lin. Three-stage heuristic algorithm for three-dimensional irregular packing problem. *Applied Mathematical Modelling*, 41:431 – 444, 2017.

[218] Y. Wu, W. Li, M. Goh, and R. de Souza. Three-dimensional bin packing problem with variable bin height. *European Journal of Operational Research*, 202(2):347–355, 2010.

[219] X.-S. Yang, S. Koziel, and L. Leifsson. Computational optimization, modelling and simulation: Past, present and future. *Procedia Computer Science*, 29:754 – 758, 2014.

[220] L. H. W. Yeung and W. K. S. Tang. A hybrid genetic approach for container loading in logistics industry. *IEEE Transactions on Industrial Electronics*, 52(2):617–627, 2005.

[221] D. Zhang, Y. Peng, and S. C. H. Leung. A heuristic block-loading algorithm based on multi-layer search for the container loading problem. *Computers & OR*, 39(10):2267–2276, 2012.

[222] X. Zhao, J. Bennell, T. Bektas, and K. Dowsland. A comparative review of 3d container loading algorithms. April 2014.

[223] J. . Zheng, C. . Chien, and M. Gen. Multi-objective multi-population biased random-key genetic algorithm for the 3-d container loading problem. *Computers and Industrial Engineering*, 89:80–87, 2015.

[224] J. Zhihong, I. Takahiro, and O. Katsuhisa. The Three-Dimensional Bin Packing Problem and Its Practical Algorithm. *JSME International Journal Series C Mechanical Systems, Machine Elements and Manufacturing*, 46(1):60–66, 2003.

[225] W. Zhou-jing and L. Kevin W. Layer-layout-based heuristics for loading homogeneous items into a single container. *Journal of Zhejiang University SCIENCE A*, 8(12), 2007.

[226] E. Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications*. PhD thesis, Zurich, Switzerland, November 1999.

[227] E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8:173–195, 2000.

[228] E. Zitzler and S. Künzli. Indicator-Based Selection in Multiobjective Search. In *VIII Conference on Parallel Problem Solving from Nature*, volume 3242 of *LNCS*, pages 832–842. Springer, 2004.

[229] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimization and Control*, pages 19–26, Barcelona, Spain, 2002. CIMNE.

[230] E. Zitzler and L. Thiele. An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach. Technical Report 43, Computer Engineering and Networks Laboratory (TIK), 1998.

[231] E. Zitzler and L. Thiele. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In *Parallel Problem Solving from Nature*, volume 1498 of *LNCS*, pages 292–301. Springer, 1998.

[232] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.