Instance space analysis for the generalized bin packing problem algorithms

A research project submitted in fulfillment of the requirements for the degree of

MSc IN OPERATIONS RESEARCH

in the

Department of Statistical Sciences

Author:
Funanani Netshitungulu

Supervisor:
Dr R Georgina Rakotonirainy

June 24, 2025

Declaration of Authorship

I know the meaning of plagiarism and declare that all of the work in the dissertation, save for that which is properly acknowledged, is my own.

I confirm that:

- This work was done wholly or mainly while in candidature for a Masters degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Name: Funanani Netshitungulu Date: 14 February 2025

Signed: Signed by candidate

University of Cape Town

Abstract

Faculty of Science
Department of Statistical Sciences

Instance space analysis for the generalised bin packing problem algorithms

Funanani Netshitungulu

under the supervision of
Dr. R. Georgina Rakotonirainy

It is well known that the bin packing problem is one of the most studied combinatorial optimization problems. In the generalised bin packing problem, the objective is to pack a selected set of profitable non-compulsory items with all the compulsory ones into a set of bins such that the resulting packing cost is minimised. The total cost is given by the difference between the cost of the selected bins and the total profit of the loaded items. This type of problem is encountered in logistics, mainly in the transportation industry which has grown massively over the years. In this thesis, six improved heuristics are proposed to tackle this problem. The aim is to investigate the upper bound solutions provided by such heuristic approaches to the problem. An Instance Space Analysis is also applied to test the efficiency and effectiveness of the algorithms in respect of the problem instance space. In particular, the relationship between the problem instance features and the algorithm performance is studied. The results indicate that the chosen features are able to explain the difficulty of the problem instances, highlighting the strengths and weaknesses of the various algorithms. This work contributes to the advancement of research in the context of packing problem instance space analysis.

Acknowledgments

This thesis is a result of almost two years of research undertaken at the University of Cape Town, for the degree of 'Masters of Science in Operations Research'. Throughout this journey, I received a lot of support and assistance, which I would like to acknowledge.

I would like to thank my supervisor, Dr. R. Georgina Rakotonirainy, whose generous help throughout the duration of this project has been immense. This was not an easy task, and I highly appreciate your guidance throughout and for providing me with the relevant resources to fast track my progress. Thank you.

CONTENTS

LIST OF ACRONYMS

GBPP: Generalized Bin Packing Problem
BPP: Bin Packing Problem
VSBPP: Variable Size Bin Packing Problem
VCSBPP: Variable Cost and Size Bin Packing Problem
KP: Knapsack Problem
MKP: Multiple Knapsack Problem
MKPI: Multiple Knapsack Problem with Identical capacities
MVCSBPP: Multi-Period Variable Cost and Size Bin Packing Problem
MVCSBPP-AC: Multi-Period Variable Cost and Size Bin Packing Problem with assignment cost
BPP-IF: Bin Packing Problem with conflicts and Item Fragmentation
ISA: Instance Space Analysis

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1

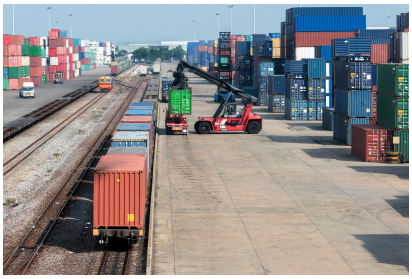
INTRODUCTION

The logistics industry has experienced significant growth in recent years, with companies, particularly those in transportation, consistently confronted with the challenge of moving goods across expansive distribution networks, whether between warehouses or directly to customers. The costs associated with these movements represent a substantial portion of both distribution and operational expenses for these organizations. Distribution itself is a critical component of the supply chain, as highlighted in prior studies [63]. Regardless of the mode of transportation—be it road, air, sea, or rail—goods are continuously being transported, with transportation serving as the primary revenue stream for most companies in the sector. Consequently, the process of loading goods for distribution has become a crucial factor influencing corporate profitability, thereby giving rise to Bin Packing Problems (BPPs). These problems are prevalent across a variety of real-world contexts, as exemplified in Figure 1.1.

(a) road freight [17].

(b) air freight [52].

(c) ocean freight [1].



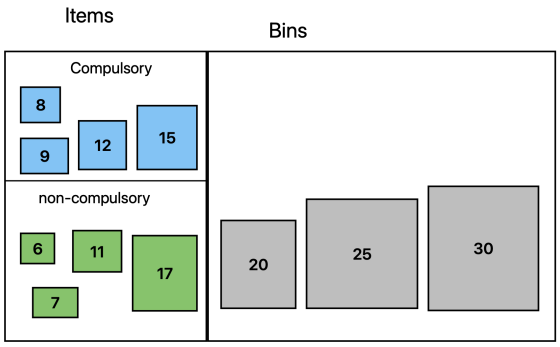
(d) rail freight [46].

Figure 1.1: Examples of bin packing problems seen in real-world application.

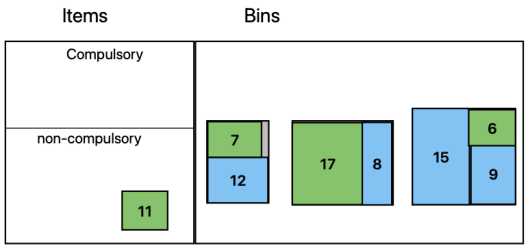
1.1 Problem description

The BPP is a type of packing problem in which given a list of items of various sizes and a finite number of bins with a fixed capacity, all items have to be packed into a minimum number of bins [45]. This study focuses on a specific variant of BPPs, namely the Generalized Bin Packing Problem (GBPP). The GBPP involves the efficient packing of a set of items into a collection of bins in a manner that maximizes profitability. The items are categorized into two types: compulsory and non-compulsory, each characterized by both volume and profit. Similarly, the bins are defined by their volume and associated cost. The objective is to load the compulsory items along with a subset of the most profitable non-compulsory items into the appropriate bins, thereby minimizing the total net cost [2]. The total net cost is calculated as the difference between the total cost of the bins utilized and the total profit derived from the loaded items.

An illustrative example of the GBPP, depicted in Figure 1.2, demonstrates how the packing process works, with all compulsory items being packed into bins along with a selection of profitable non-compulsory items. This type of packing problem is predominantly encountered in transportation logistics, where it was introduced to address the simultaneous consideration of bin costs and item profits, while also accounting for constraints related to bin availability, as well as the varying costs and volumes of the bins [2]. Given the significant impact of these costs on the operational expenses of companies within this sector, numerous decision support tools have been developed by researchers to mitigate distribution costs [4].



(a) Initial setting of a GBPP.



(b) Final packing of a GBPP.

Figure 1.2: An illustrative example of the GBPP with all compulsory items and a subset of non-compulsory items packed. The numbers inside the items and bins indicate their respective capacities.

1.2 Motivation of the study

Since the introduction of the GBPP, several algorithms have been developed to address the problem and its various variants. These algorithms have primarily been evaluated based on their average performance. However, even the most effective algorithms may have inherent weaknesses that are often overlooked or inadequately reported. As a result, the algorithm that performs best on average is typically adopted for all scenarios, even in cases where it may not be the most effective choice. The assumption is that, over time, the algorithm will yield satisfactory results in the long run.

But what if we could move beyond merely reporting average performance? According to Cunha *et al.* [14], determining the most suitable algorithm for a given problem instance is one of the most critical yet under-explored issues. It is conceivable that profit margins could significantly increase if the best-performing algorithm could be complemented by other algorithms that excel in areas where the primary algorithm falls short. This study aims to address this gap by applying an Instance Space Analysis (ISA) approach to GBPP algorithms for the first time. The goal is to gain deeper insights that extend beyond average performance metrics. As a result, this approach enables more precise recommendations regarding which algorithms should be employed for specific instances. Instead of relying on the algorithm with the best overall average performance, we propose selecting the algorithm that is best suited to each individual problem instance.

1.3 Project Aim and Objectives

The widespread applicability of BPPs has garnered substantial attention, leading to the development of various solutions. Specifically, the GBPP, owing to its relatively recent introduction, still lacks highly efficient and effective packing algorithms. While literature on the GBPP remains somewhat limited, there has been a noticeable increase in research efforts, with several variants of the problem having been proposed in recent years.

In scenarios where the number of items and bins is small, the problem can be easily solved using linear programming and leveraging solvers or through brute force algorithm until an optimal solution is identified. However, as the problem becomes more complex—characterized by an increase in the number of items and bins—the difficulty of finding a solution escalates, becoming more costly and time-consuming, this is due to the nature of the problem being NP-hard [6]. Consequently, previous solutions to packing problems have increasingly relied on heuristics, which have also been applied to the GBPP. In this context, heuristics refer to a set of rules or strategies used in subsequent studies to develop algorithms that expedite the packing process, delivering near-optimal solutions to problems involving large sets of items and bins where linear programming approaches become impractical.

The objective of this thesis is to contribute to the heuristic solution methods proposed for the GBPP, specifically addressing the computational challenges posed by linear programming techniques, which are; large computational times, excessive costs and failing to solve the problem to optimality due to complexity. Existing solutions from the literature will be examined as a foundation for suggesting potential improvements. Furthermore, the performance of the proposed solutions will be compared against benchmark heuristics in terms of solution quality.

This comparison will involve assessing the total net cost produced by each algorithm across a range of benchmark instances, and statistical tests will be conducted to evaluate whether the proposed algorithms offer improvements over the benchmark solutions. Additionally, the proposed solutions will be tested over a larger instance space to assess the relative strengths and weaknesses of each algorithm. This approach aims to go beyond simply reporting average performance, marking the first time that the strengths and weaknesses of heuristics for the GBPP will be systematically evaluated. Insights regarding the performance of individual algorithms will be drawn through the application of an ISA method, providing a more nuanced understanding of each algorithm's performance in different contexts.

The aforementioned aims are accomplished by pursuing the following objectives:

i. To conduct a literature review on the GBPP and its variants with respect to solution approaches, exact algorithms and mainly heuristics, typically employed to solve the GBPP and similar problems.

ii. To conduct a literature review on ISA and how it has been applied to bin packing problems.

iii. To collect a set of benchmark instances of the GBPP in order to compare solution approaches reviewed in objective *i*,

iv. To create additional problem instances to use in feature selection as per solutions reviewed in objective *ii*.

v. To implement the reviewed heuristics in pursuit of objective *i* on the same platform and apply them to the data instances identified in fulfillment of objective *iii*.

vi. To propose possible improvements to existing heuristics to solve the GBPP.

vii. To conduct an ISA on all algorithmic approaches under consideration in terms of solution qualities they yield.

viii. To perform an appraisal on the contributions made during the study, and to recommend future work of interest which may be pursued in future.

1.4 Project Structure

This thesis contains eight chapters in addition to the current introductory chapter, all grouped into three parts. The first part, comprised of two chapters, is dedicated into reviewing literature on the GBPP and the solutions approaches towards solving the problem, and instance space analysis, and how it has been used in the context of packing problems. The second part also consists of two chapters which contains newly proposed heuristic algorithms and how instance space analysis was applied using these algorithms. The third part consists of two chapters dedicated to evaluating the performance of the algorithms and how effective they are. The final chapter gives a summary and appraisal of contributions of this thesis, together with identifying more other interesting areas of research for follow up investigation.

The scope of the problem addressed in this thesis is thoroughly detailed in Chapter 2. This chapter begins with an introduction to the GBPP and its various problem variants. It

commences by outlining the most widely recognized typology within the domain of packing problems. Subsequently, a concise review of solution methodologies for the GBPP, along with related problems, is provided. The solution approaches discussed include both exact methods and heuristic strategies. Finally, the chapter concludes with a description of the packing problem and the specific solution techniques considered in this thesis.

Chapter 3 presents a comprehensive review of the ISA algorithm. This chapter outlines the general framework of ISA and its constituent components. It further explores instances where ISA has been previously applied to packing problems. The chapter concludes with a specific focus on how ISA is utilized in the context of this thesis. In Chapter 4, the six proposed improved algorithms are discussed in detail. Pseudo-codes are provided to facilitate a clear understanding of the operation of each algorithm. Additionally, figures illustrating the results of the algorithms are included to enhance comprehension of their performance.

Chapter 5 introduces the meta-data used throughout the thesis in the context of ISA. The chapter outlines how the ISA framework was applied to the GBPP, detailing the configuration and implementation of the algorithm. Chapter 6 presents the results obtained from the heuristic approaches, focusing on the percentage gap between the lower and upper bounds. These results are displayed through boxplots and tables, providing a visual comparison of the performance of the heuristics. Additionally, the chapter reports on the net cost results and includes tables from post-hoc statistical tests, all conducted at the 95% confidence level. These findings offer a comparison of the average performance of the heuristic algorithms.

In Chapter 7, the results from applying ISA to the GBPP are presented in graphical form. The graphs provide a deeper understanding of how each algorithm performs on individual instances across the instance space, offering insights that go beyond average performance metrics. This chapter also includes performance tables and presents updated ISA results based on additional generated instances. These updated results are displayed in graphs and further extend the insights obtained from the initial ISA analysis. Chapter 8 discusses the results in detail, including the average net cost results from the heuristics and the insights derived from the ISA approach. Finally, Chapter 9 concludes the thesis with a summary of the work presented, an appraisal of the contributions made, and suggestions for future research directions.

Part I
Literature review

CHAPTER 2

THE GENERALISED BIN PACKING PROBLEM

This chapter provides a thorough overview of the state-of-the-art typology in GBPP research, presenting the key algorithms and extensions that have been developed to address its various facets.

2.1 Types of packing problems

The GBPP addressed in this research is classified within the broader family of Cutting and Packing (C&P) problems, a category that has been extensively studied over the past century. The problem has been referred to by various names in the literature, including the cutting stock problem [24, 25, 36], the trim loss problem [21], and bin packing problems [19, 32]. The first comprehensive typology of these problems was proposed by Dyckhoff [20]. In 2007, Wascher *et al.* [64] proposed an enhanced typology, which remains the standard in the field.

A summary of Wascher's typology is presented in Table 2.1. Their categorization includes five key characteristics. *Dimensionality:* This characteristic distinguishes between one-, two-, and three-dimensional packing problems, acknowledging that variants beyond three dimensions can also be classified, though they are treated as special cases. *Kind of Assignment:* This refers to whether the problem involves input (value) minimization or output (value) maximization. In input minimization, there are sufficiently large objects to accommodate all small items, while in output maximization, the large objects are insufficient to accommodate all small items, meaning only a subset of small items can be packed. *Assortment of Small Items:* Small items can be categorized as either identical, weakly heterogeneous (where subsets of items are identical), or strongly heterogeneous (where there are few or no identical items). *Assortment of Large Objects:* This characterizes the large objects as either a single object or multiple objects. In the case of a single large object, the dimensions may be fixed or variable. For multiple large objects, they can be identical, weakly heterogeneous (subsets are identical), or strongly heterogeneous (few identical objects). Finally, *Shape of Small Items:* Items may be regular (e.g., rectangular, circular) or irregular in shape.

Kind of assignment		Output maximisation			Input minimization		
Assortment of small items		Identical	weakly heterogeneous	Strongly heterogeneous	Arbitrary	weakly heterogeneous	Strongly heterogeneous
Assortment of large objects	Identical	–	Multiple Identical Large Object Placement Problem (MILOPP)	Multiple Identical Knapsack Problem (MIKP)	–	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	–	Multiple Heterogeneous Large Object Placement Problem (MHLOPP)	Multiple Heterogeneous Knapsack Problem (MHKP)	–	Residual Cutting Stock Problem (RCSP)	Residual Bin Packing Problem (RBPP)
	One large object	Identical Item Packing Problem (IIPP)	Single Large Object Placement Problem (SLOPP)	Single Knapsack Problem (SKP)	Open Dimensional Problem (ODP)	–	–
Type of C&P problem		IILP	PP	KP	ODP	CSP	BPP

Table 2.1: Types of C&P problems according to the typology of Wascher *et al.* [64]

This typology leads to six distinct types of C&P problems, including the Identical Item Packing Problem (IIPP), Placement Problem (PP), Knapsack Problem (KP), Open Dimensional Problem (ODP), and Cutting Stock Problem. The GBPP considered in this work falls under the BPP category and generalizes other problems, such as the KP.

Based on the assortment of large objects, BPP variants can be classified as Single Bin Size Bin Packing Problem (SBSBPP), Multiple Bin Size Bin Packing Problem (MBSBPP), or Residual Bin Packing Problem (RBPP). The SBSBPP involves strongly heterogeneous small items packed into a set of identical large objects, with the objective of minimizing the number of bins used. The MBSBPP involves strongly heterogeneous small items packed into a set of weakly heterogeneous large objects [29, 34]. The RBPP involves strongly heterogeneous small items packed into a set of strongly heterogeneous large objects, with the objective of minimizing the total cost of the large objects [4].

The KP is characterized by the packing of a strongly heterogeneous assortment of small items into a given set of large objects. Relevant variants of the KP include the SKP [7, 15, 23, 42], which involves a single large object and aims to maximize the profit of the items; the MHKP [8, 43], which involves a strongly heterogeneous assortment of large objects; and the MIKP [8], which involves multiple identical large objects.

2.2 The GBPP and its variants

Let the following parameters used in the formulation of the GBPP be defined as follows:

I	the set of items.
I^c	the subset of compulsory items.
I^{nc}	the subset of non-compulsory items.
J	the set of available bins.
T	the set of bin types.
U	the total number of available bins.
L_t	the minimum number of a certain bin type that may be selected.
U_t	the maximum number of a certain bin type that may be selected.
C_j	the cost of bins.
p_i	the profit of items.
W_j	the bin volume.
w_i	the item volume.
$\sigma(j)$	the type of bin.

Table 2.2: Parameters and indices used in the mathematical formulation of the GBPP and their descriptions.

Consider the following decision variables:

- Bin selection variables y_j equal to 1 if bin $j \in J$ is used, 0 otherwise.
- item-to-bin assignment variable x_{ij} equal to 1 if item $i \in I$ is loaded into bin $j \in J$, 0 otherwise.

Baldi *et al.* [2] formulated the problem as an integer programming model, with the objective of minimizing the net cost subject to constraints based on the parameters presented in Table 2.2.

Min:

$$\sum_{j \in J} C_j y_j - \sum_{j \in J} \sum_{i \in I^{nc}} p_i x_{ij} \tag{2.1}$$

S.t:

$$\sum_{i \in I} w_i x_{ij} \leq W_j y_j, \quad \forall j \in J, \tag{2.2}$$

$$\sum_{j \in J} x_{i,j} = 1, \quad \forall i \in I^c, \tag{2.3}$$

$$\sum_{j \in J} x_{i,j} \leq 1, \quad \forall i \in I^{nc}, \tag{2.4}$$

$$\sum_{j \in J : \sigma(j) = t} y_j \leq U_t, \quad \forall t \in T, \tag{2.5}$$

$$\sum_{j \in J : \sigma(j) = t} y_j \geq L_t, \quad \forall t \in T, \tag{2.6}$$

$$\sum_{j \in J} y_j \leq U, \tag{2.7}$$

$$y_j \in \{0, 1\}, \quad \forall j \in J, \tag{2.8}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in I, j \in J. \tag{2.9}$$

In this formulation, items are assigned to bins with an objective of minimizing the net cost, given by the difference between total cost of used bins and the profit of non-compulsory items (2.1). The profit of compulsory items was not considered as it is constant and guaranteed.

Constraint (2.2) ensures that the volume of items in a bin cannot exceed the bin's volume. Constraint (2.3) states that all compulsory items should be packed into the bins while constraint (2.4) indicates that a subset of compulsory items can be packed, and some can be left unpacked. Constraints (2.5) and (2.6) bound the number of bins for a certain bin type, this could be company dependent with the company specifying the number of a certain bin type available for use. Constraint (2.7) states that the total number of used bins cannot exceed the total number of bins available for use. Finally, constraints (2.8) and (2.9) state that the selection of a bin and whether an item is assigned to a particular bin are binary. Thus, a bin can either be selected or not selected ($y_i = 1$ if bin is selected, 0 otherwise) and an item can either be assigned to that bin or not assigned ($x_{ij} = 1$ if item i is assigned to bin j).

Since its introduction, several extensions of the Generalized Bin Packing Problem (GBPP) have been proposed in the literature. In 2014, Baldi *et al.* [4] introduced an extension known as the Stochastic Generalized Bin Packing Problem (S-GBPP). This extension introduces a random term to the profit of items, representing fluctuations in profit due to handling costs when packing items into bins. Such random fluctuations could occur in scenarios like mishandling of a delivered package, where a customer complaint and compensation would result in a reduced profit for the item. Aside from this addition, the problem retains all the characteristics of the GBPP, with the primary objective being to maximize the net profit, defined as the difference between total profit and the total cost of bins used.

Another significant extension is the Generalized Bin Packing Problem with Bin-Dependent Item Profits (GBPPI), proposed by Baldi *et al.* [3]. This variant addresses more realistic performance indicators and incorporates mixed objective functions. The mixed objectives include (i) minimizing the total cost of the bins used, and (ii) maximizing the bin-dependent profit of the selected items. In this extension, the profit associated with an item is contingent upon the bin in which it is packed, reflecting real-world scenarios where the value of packing an item in a specific bin might vary.

In 2022, Ding *et al.* [18] introduced the Generalized Extensible Bin Packing Problem with Overload Cost (GEBPPOC). In this problem, the total size of the items packed into a bin can exceed the bin's capacity, leading to overload costs. The objective is to minimize the total cost, which includes both the fixed costs of the bins and the additional overload costs incurred from exceeding bin capacities.

More recently, Rodrigues *et al.* [48] proposed the Generalized Bin Packing Problem with Incompatible Categories (GBPPIC). This extension addresses the issue of incompatible categories, where certain types of items cannot be transported together. The goal in the GBPPIC is to optimally assign deliveries of different products to a set of delivery vehicles, minimizing the number of vehicles required while respecting the constraints imposed by incompatible item categories.

2.3 Solution approaches

Although the literature on the GBPP is very limited due to its recent introduction, there have been many solutions proposed to handle such problems. This is because the roots of the solutions for the GBPP are based on the BPP which has a lot of research around it. The solutions proposed have been either exact methods or heuristic methods. The exact methods, which uses solvers to find the optimal solution to the problem are based on mathematical formulation. However, as the problems get more complex, these methods tend to fail or take too long to compute the results. Heuristic methods find near-optimal solutions to the problem within a reasonable time frame, as such, they become practical to use in large problem settings. This section provides a brief overview of these solutions.

2.3.1 Exact methods

Exact methods are usually based on a mathematical programming model, and are guaranteed to find the optimal solution to a GBPP problem instance. Although these methods solve the problem to optimality, they are deemed impractical and in solving realistically sized problem instances [45]. This is because of the NP-hard nature of the problem.

The original GBPP literature utilized two exact methods to compile efficient lower bound solutions, namely the assignment formulation and the set covering formulation [2]. The assignment formulation provides a good starting point to solving the problem. In this formulation, the objective consists of minimizing the net cost subject to volume and packing constraints (it involves a polynomial number of variables and constraints). This formulation was used to derive the lower bounds which were also required for the purpose of this thesis. In this case, the problem was treated as an aggregate knapsack problem, and dropping some

of the constraints from the original formulation in Section 2.2 resulted in the following model:

Min:

$$\sum_{j \in J} C_j y_j - \sum_{i \in I^{nc}} p_i x_i \tag{2.10}$$

S.t:

$$\sum_{i \in I^c} w_i + \sum_{i \in I^{nc}} w_i x_i \leq W_j y_j, \tag{2.11}$$

$$\sum_{j \in J} y_j \leq U, \tag{2.12}$$

$$y_j \in \{0, 1\}, \quad \forall j \in J, \tag{2.13}$$

$$x_i \in \{0, 1\}, \quad \forall i \in I. \tag{2.14}$$

The set covering formulation provides another lower bound solution to the problem. It uses a column generation method to pack items into bins. For the GBPP, this formulation was based on feasible loading patterns for the bins, which is defined as a set of items that may be loaded into a bin, while satisfying all dimension restrictions and accommodation rules [2]. Using this formulation, the feasible loading pattern was described as a vector with entry 1 if an item is packed into the pattern or 0 otherwise. The objective was set to be minimizing the total cost of selected bin loading patterns. The generated pattern addresses feasibility of loading the items, while the model only focuses on the optimality of the combination of patterns. Rodrigues *et al.* [48] also used these two exact methods, although adjusted for incompatible categories for their variant of the GBPP.

In 2012, Baldi *et al.* [4] developed a deterministic approximation for the S-GBPP. Using this method, they were able to easily calculate the random profit oscillations term by calculating the maximum random profit of each item. They proved that the random profit can be modelled as a double exponential probability distribution, which made it easy to calculate. Thus, their model is practical in situations where the profit of an item is not always fixed after packing the item and can randomly fluctuate depending on how the items are handled. The objective of their formulation was to maximize the total net profit, which is the difference between expected profit of loaded items and total cost of used bins.

2.3.2 Heuristic methods

In the context of the GBPP, heuristics are algorithms that aim to pack a set of compulsory and non-compulsory items into bins following a predefined set of rules, ensuring that the solutions remain feasible. Baldi *et al.* [2] introduced two key heuristic variants based on two basic algorithms: First Fit Decreasing (FFD) and Best Fit Decreasing (BFD). These variants are referred to as U-FFD and U-BFD in this thesis.

The U-FFD algorithm operates by first sorting the items in non-increasing order of volume, known as the Sorted Item List (SIL), and sorting the bins in increasing order of capacity, known as the Sorted Bin List (SBL). The algorithm initially considers used bins, starting with the empty list of used bins. For compulsory items, the algorithm searches for the first used bin in which the item can fit. If no used bins can accommodate the item, a new bin is opened. For non-compulsory items, the algorithm prioritizes fitting the item into a used

bin. If this is not possible, the profitability of packing the item is evaluated, and a new bin is opened if it is profitable. Specifically, a new bin is opened only if the total profit (item profit plus remaining items' profit in the SIL) exceeds the cost of the bin, provided the combination of items can fit in the bin. Items that do not meet this condition are discarded.

The U-BFD algorithm shares many similarities with U-FFD but differs in how bins are selected. Instead of fitting an item into the first available bin, items are packed into the best bin, which refers to the bin that leaves the least residual space after packing. For compulsory items, the best bin is the first bin in the sorted list of bins, while for non-compulsory items, the process mirrors that of U-FFD, selecting the best bin based on the same profit considerations.

Baldi *et al.* [3] introduced two other heuristics, the Best Profitable (BP) and Best Assignment (BA), both of which are based on the U-BFD algorithm but extend the concept of the best bin. The key difference between BP and BA lies in how new bins are opened. The BP algorithm opens a new bin if the combined profit of an item and any additional items that can fit into the bin exceeds the bin's cost. The BA algorithm, on the other hand, opens the bin that maximizes the profit of the item being packed. Both approaches aim to optimize the packing of items with consideration for bin-dependent profits, the GBPPI problem.

In addition to the heuristics, Baldi *et al.* [3] proposed two metaheuristics: the Greedy Adaptive Search Procedure (GASP) and the Model-Based Matheuristic (MBM). The GASP algorithm involves generating multiple feasible solutions and using them as starting points for further iterations. The starting points are generated using the BP or BA. In each iteration, a 1 to 1 swap is performed and a packed item is swapped with an unpacked item if the swap is feasible and profitable, continuing until no further improvements can be made or a set number of iterations is reached. The MBM algorithm operates by simultaneously solving two subproblems—minimizing the total cost and maximizing item profits—and combining the results, ensuring no duplicates in the solutions.

In 2019, Crainic *et al.* [13] proposed seven heuristic algorithms to solve the VCBPP-AC and the MVCBPP-AC. Three of these algorithms, HS1, HS2, and HS3, focus on packing items based on their cost contributions and the residual space in bins. The HS1 algorithm prioritizes items with the highest cost-to-residual capacity ratio, the HS2 algorithm selects bins based on the lowest cost-to-capacity ratio, and the HS3 approach is similar to HS2, but it adds the cost of items to the bin cost when selecting bins. The other four algorithms proposed were based on the multi-period setting of the problem, which is different from the setting considered in this thesis.

To address the GEBPPOC, Ding *et al.* [18] introduced two heuristic algorithms: Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). These algorithms modified traditional solution representations and operators, adjusting methods such as the construction of the initial solution and setting termination criteria to efficiently solve the GEBPPOC and deliver high-quality results.

For solving the GBPPIC, Rodrigues *et al.* [48] proposed the Adaptive Large Neighbourhood Search (ALNS) metaheuristic. Initially designed for vehicle routing problems [43], the ALNS has been adapted for various problem types, including project scheduling and lot-sizing [62]. Its application to the GBPPIC is particularly suitable, as it helps to efficiently address the

issue of incompatible item categories while minimizing the number of vehicles required. In this setting, the ALNS starts with multiple initial solutions, multiple destroy and repair methods used in order to improve the solution after each iteration. The destroying method and repairing method respectively to use is chosen based on the probability of success in improving solutions in previous iterations [48].

2.4 Focus of this thesis

In this thesis, the focus is on the one dimensional setting of the GBPP in which the only characteristic (dimension) of items is volume. In this problem, there are sufficiently large objects (bins) in which the small items can be assigned, the small items are strongly heterogeneous, thus, there are only a few identical items in an instance if any. There are several large objects which are weakly heterogeneous (bins only consist of at most five unique capacities on individual instances). The items have regular shape, with the assumption that it is rectangular. Table 2.3 shows the characteristics of the GBPP considered in this work in terms of Wascher *et al.*'s typology.

Criteria	GBPP setting
Dimensionality	one-dimensional (1D)
Kind of assignment	input (value) minimisation
Assortment of small items	Strongly heterogeneous
Assortment of large objects	Weakly heterogenous
Shape of small items	Regular (boxes)

Table 2.3: A representation of the problem setting investigated in this work according to the Wascher *et al.*'s typology [64].

In terms of the solution approaches, the focus is on application of heuristics to solving the GBPP. Eight algorithms are considered and compared in terms of the objective value that they produce. After which, the objective values will be compared to the lower bounds in terms of the percentage gap. The lower bounds were computed using the Baldi *et al.*'s assignment model presented in Section 2.3, which was solved to optimality as an aggregate knapsack problem.

CHAPTER 3

THE INSTANCE SPACE ANALYSIS FRAMEWORK

It is a fact that assessing algorithm performance objectively is challenging. As argued by Hooker [30], an algorithm can perform well on a given or chosen set of data instances but there is no inherent guarantee that it will do so on different instances. Liu *et al.* [40] also stated that one cannot simply consider the average performance of an algorithm over a possibly unrepresentative set of problem instances. The ISA was proposed by Smith-Miles *et al.* [53] in an attempt to alleviate such bias. The idea behind ISA is to project a given complex problem into a 2D instance space in which algorithms can be tested and compared to gain valuable insight into their performance. In this chapter a detailed framework of the ISA is presented and its application in the context of packing problems. At the end, an outline of the focus of this thesis in terms of ISA is discussed.

3.1 General framework of ISA

The ISA approach maps problem instances onto a two-dimensional space, referred to as the 'instance space', where each instance is represented as a distinct point. This mapping is achieved by calculating specific attributes of the problem instances, known as 'features'. The goal is to select features that effectively capture the difficulty of each instance for the algorithms being evaluated. The projection of problem instances onto the instance space enables the visualization of these instances, allowing for a clearer understanding of where the algorithms excel or struggle. This process provides deeper insights into algorithm performance, extending beyond the simple reporting of average-case performance.

The ISA by Smith-Miles *et al.* [53] builds upon the foundational concept of the Algorithm Selection Problem (ASP), first proposed by Rice in 1976 [47]. Rice's ASP framework aimed to establish a mapping between algorithms and a set of performance metrics, enabling the selection of the most appropriate algorithm for a given problem instance. The ISA framework further refines this concept by seeking to map each algorithm's performance onto the problem instance space in a manner that optimizes the selection process. Additionally, the framework allows for the extraction of the algorithm's 'footprint', a representation of the algorithm's

behavior across different instances and performance measures, as depicted in Figure 3.1.

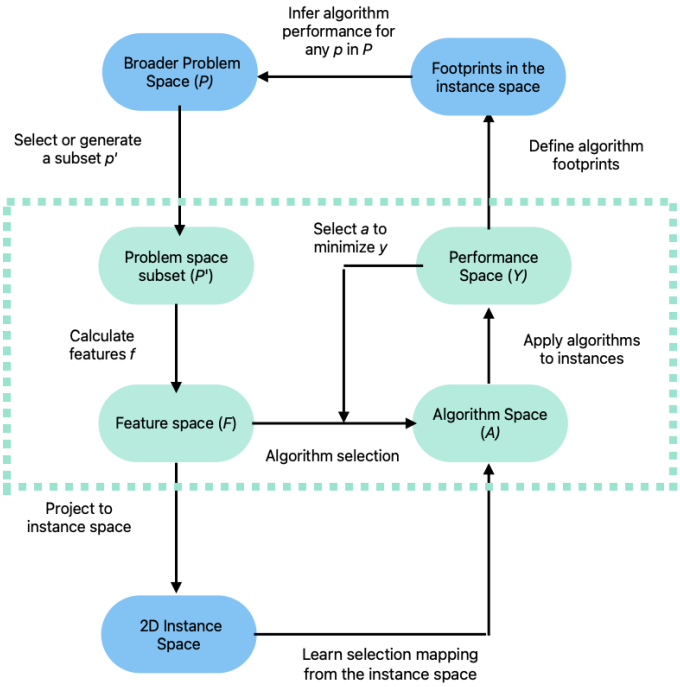


Figure 3.1: A representation of the ISA framework by Smith-Miles [53], extending the original ASP framework by Rice [47], shown in the dotted box.

3.1.1 Broader problem space

A comprehensive selection of instances is essential, which can be accomplished by gathering benchmark problem instances documented in existing literature. If the available instances are insufficient, new ones can be generated using existing benchmark generators. For packing problems, a substantial collection of instances is already available, such as those found in the ESICUP repository [22]. However, these instances often exhibit significant similarity, which may limit the diversity needed across measurable features. As a result, additional instances may need to be generated to expand the boundaries of the instance space. Increased variation in problem characteristics can be achieved by utilizing problem generators capable of producing a wide range of instances with varying parameters under controlled conditions. An example of a packing problem generator is the 2DCPackGen by Silva *et al.* [51].

3.1.2 Subset of the problem space

It is important to note that assessing the diversity of instances across the instance space involves two key considerations: instance dissimilarity and algorithmic discrimination. The former key refers to the need for the instances to cover a broad and representative region within the instance space. The latter one ensures that the instances are able to distinguish between the behaviors and relative performance of each algorithm within the instance space. Thus, the process of generating diverse instances is inherently iterative, as it requires continuous refinement to balance both dissimilarity and discrimination. In addition, a subset of the problem space (P') can be considered for computational efficiency.

3.1.3 Feature space

The feature space contains all relevant features that best characterise the problem instances under consideration and that can used to assess the algorithm performance. The selected features should be chosen in such a way that they reveal the varying complexities of the problem instances, capture any known structural properties, and establish connections between the features and the strengths and limitations of different algorithms. A variety of feature selection techniques have been proposed in the literature, including supervised feature selection methods and principal component analysis [26].

3.1.4 2D instance space

The feature space is often high-dimensional due to the large number of features involved. To facilitate better analysis, these features are typically projected into a 2D instance space. This projection enhances the visualization of the distribution and diversity of the problem instances, making it easier to identify areas where each algorithm performs well or poorly. Additionally, it highlights regions in the instance space where generating more instances could provide further insights. These regions may include areas that are either underrepresented by existing instances or where no algorithm exhibits a clear superiority. Furthermore, this approach assists in evaluating the adequacy of the selected features. Specifically, the features should effectively predict algorithm performance by capturing the strengths and weaknesses of the algorithms.

3.1.5 Algorithm space

The algorithm space encompasses all the algorithms under investigation. As discussed in Section 2.3, both exact and heuristic algorithms can be employed to solve packing problems. While other solution approaches are available in the literature, they fall outside the scope of this work.

3.1.6 Performance space

The effectiveness of the various algorithms is assessed using a standard performance measure commonly applied in the field of packing problems. While various performance evaluation methods exist in the literature, the most widely used computational metric for assessing the performance of a packing algorithm is the relative difference between the packing solution produced by the algorithm and the optimal solution or lower bound to the problem [31]. This metric is typically expressed as a percentage gap or ratio. Another alternative performance measure is packing time efficiency, which refers to the time taken by the algorithm to find a packing solution for the problem [45]. This computation time is typically measured by tracking the time during the algorithm's execution.

3.1.7 Footprints in the instance space

Each algorithm can be evaluated in terms of its strengths and weaknesses relative to the instance space. The region within the instance space where an algorithm is expected to perform well is referred to as its footprint [58]. Several methods exist for measuring the relative size of an algorithm's footprint. One example is the ratio of the area of the convex hull

formed by the points where the algorithm demonstrates good performance to the area of the hull encompassing all proposed instances, as suggested by Smith-Miles [58]. Additionally, the effectiveness of an algorithm can be arbitrarily defined, such as by considering an algorithm as "good" if it consistently achieves a packing solution within a certain percentage of the known optimal solution or lower bound.

3.2 MATILDA

An online toolkit called MATILDA (Melbourne Algorithm Test Instance Library with Data Analytics) [41] was proposed for the purpose of ISA [53] as described in Section 3.1. The platform was developed to assist in testing algorithm performance objectively, and to visualize where algorithms perform best or poorly. This platform is used in this work and its components are detailed in this section.

After creating an account on MATILDA, one can proceed to perform algorithm analysis, either for problems that are already in the library, of which the metadata, description of the problem, and the results are given, or on a custom problem, which could be any optimisation problem in which algorithms or models are being compared (see Figure 3.2). The name of the problem under consideration must be inserted (e.g GBPP, BPP), and a brief description of the problem should given for presentation purposes. In the next step, the meta-data, which is compiled following specified submission guidelines, containing the instances, features, and results from each algorithm is uploaded, as shown in Figure 3.3.

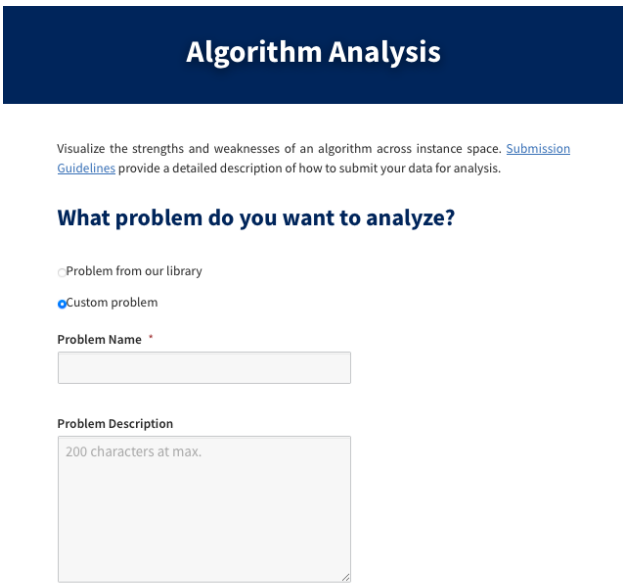


Figure 3.2: The first step of performing algorithm analysis on MATILDA.

The optimization criteria is also specified, which could either be a minimization or a maximization problem. as well as the performance criteria, which specifies how good performance is defined. The criteria could either be absolute, which says an algorithm performs good is it performs below a specified misclassification rate, or relative, if it performs within a certain percentage of the best algorithm. This percentage is known as the performance threshold and is specified together with the label of the performance metric as shown in Figure 3.3. The advanced settings can be left untouched in most cases. The beta threshold (the fraction of

algorithms that must perform good on an instance for it to be considered easy) states that an instance is easy if at least a certain percentage of the algorithms perform good on it.

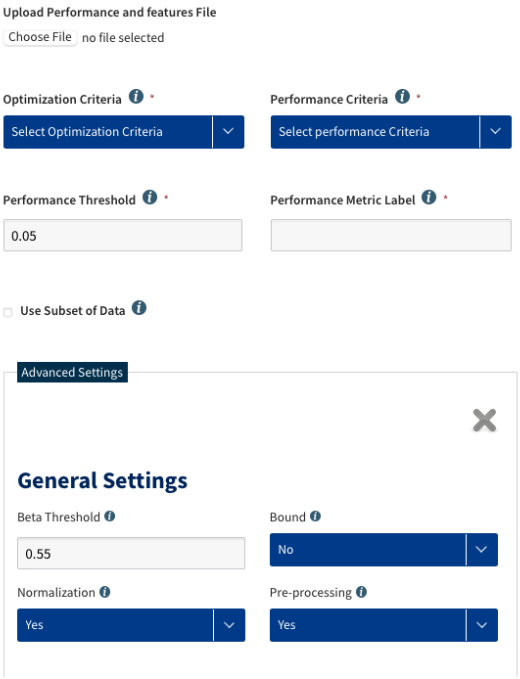


Figure 3.3: The second step of performing algorithm analysis on MATILDA.

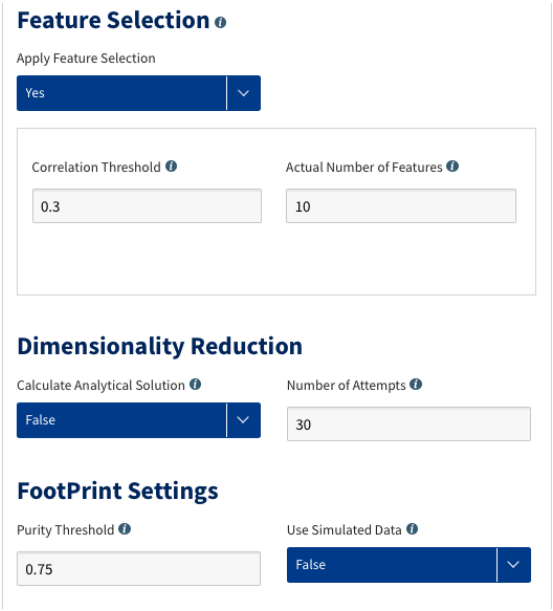


Figure 3.4: The third step of performing algorithm analysis on MATILDA.

The third step is depicted in Figure 3.4. It starts with specifying feature selection settings, where one can choose to apply feature selection, which selects only important features from the total features in the meta-data, of which the number of features are also specified. The correlation threshold specifies the minimum number of correlation accepted between features, to eradicate redundancies.

The dimensionality reduction settings specifies the number of iterations attempted before a numerical solution is reached. One can also specify whether an analytical solution should

be used or only the numerical solution to be considered, with the recommended being the numerical solution, shown in Figure 3.4. Footprint settings specify whether simulated data can be used in svm footprints or not, which helps with the final prediction of which algorithm to use in which region of the instance space. And the purity threshold specified as the minimum percentage purity required for a section of a footprint.

Support Vector Machine (SVM)[1] tuning and instance boundary settings are the last to be specified, in most cases, these can be left untouched, with the most important being the use of weights, in this case, one specifies whether weighted classification is applied or not (see Figure 3.5). Finally, the metadata is submitted, and MATILDA performs the algorithm analysis, in which it trains all the algorithms/models, and the results are saved on the user dashboard after completion.

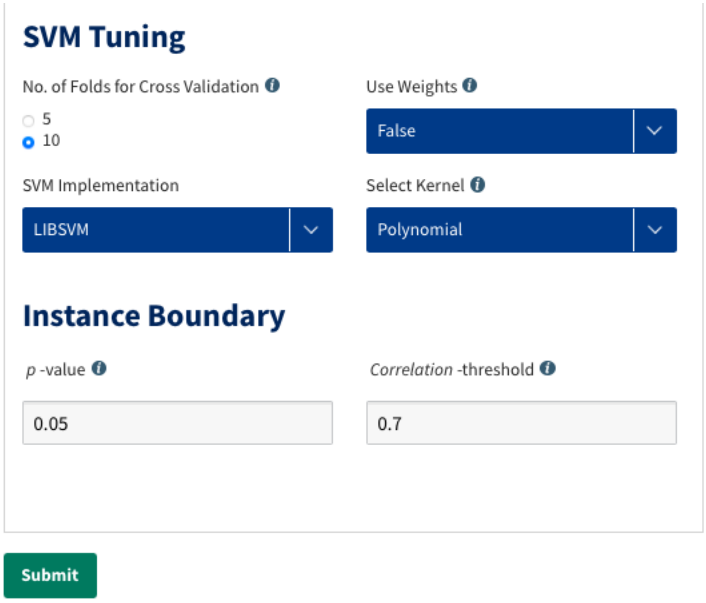


Figure 3.5: The final step of performing algorithm analysis on MATILDA.

3.3 Application of ISA in the context of packing problems

The ISA has been applied to many problems since its introduction, including graph coloring [53, 56, 59], continuous black-box optimization [54], combinatorial optimizations problems [38, 39, 53, 57], supervised classification [27], time series forecasting [35] and the Rotating Workforce Scheduling Problem [37]. Its application to the context of C&P problems is reviewed in this section.

In 2020, Liu *et al.* [40] applied the methodology of ISA to BPPs. The setting of the problem included two types of BPP, the 1DBPP and 2DBPP. Using existing problem instances from the literature and newly generated instances, they tested the a variant of the FFD, a variant of the BFD, Next-Fit Decreasing (NFD) and the Djang and Finch (DJD) algorithms for the 1DBPP. They found that the variant of the FFD was performing good in 81.5% of instances, while the DJD performed good in 79%. The DJD had a higher precision, at 91.4% compared to 88.2% for the variant of the FFD which meant that the DJD was the preferred algorithm

[1] A powerful machine learning algorithm often used for classification and regression tasks.

in regions where both algorithms perform good. They also tested the Hybrid First-Fit (HFF), Guillotine (GUILL), Maximal Rectangles (MAXREACTS) and Skyline algorithms for the 2DBPP. In this case, they found that the MAXREACTS performed well above the rest. It achieved good performance on 96.4% of the instances, which meant that the instances are not sufficiently challenging for it. Overall, ISA proved successful in providing good insight into where each algorithm performs good or bad in the instance space.

Smith-Miles *et al.* [57] applied the ISA to the 0-1 KP in 2021. Their study was based on Pisinger's work which questioned the difficulty of instances for KP problems. Their aim was to provide new test instances that increase the difficulty in order to help explain algorithm performance. Harder instances would assist in developing more state of the art algorithms to solve such problems, and considering the wide range of application for such problems, more better algorithms are always a good addition. The algorithms tested were the EXPKNAP, MINKNAP and COMBO. Instance space provided valuable insight to the question posed by Pisinger in his paper of "Where are the hard knapsack instances" as it showed regions in which all the algorithms were performing good and bad, and regions where the instances are considered easy or hard. After generating more harder instances, COMBO still proved to be the best performing algorithm, with MINKNAP in second and EXPKNAP performing bad everywhere across the instance space. As a result, COMBO was the recommended algorithm almost in the whole instance space, with MINKNAP having a little region where it is preferred. Although the problem was limited to only the base case of the knapsack problem, it still provided insight on which instances to consider when developing new algorithms.

In 2023, Jooken *et al.* [33] carried on the use of ISA for the 0-1 KP. Their study was more concerned with which features in particular make the KP hard to solve for state of the art solvers. In their study, they used machine learning models to predict empirical hardness of instances to solve. The aim was to compile harder features from the instances used for the 0-1 KP, which will help in the creation of more robust algorithms for such problems. This shows how much interest there is in developing improved algorithms for knapsack problems. As a result of their machine learning models, their proposed features were difficult enough to provide valuable insight that was lacking in previous literature due to instances being easy to solve. They stated that compiling their features was too computationally expensive, and this could possibly be the reason why features from the literature were proving too easy to solve. This links back to Pisinger's proposed question of "Where are the hard knapsack instances" which Smith-Miles *et al.* [57] attempted to answer. They improved on Smith-miles *et al.*'s [57] features as input in their model. They applied the ISA and proved that most of the hard instances come from their machine learning model. This was because COMBO proved to have significantly longer computational times for their instances in comparison to the rest. Although there were still some features from the literature that proved hard to solve. Thus, the instances from the literature were still good enough to provide insight. The new proposed features provided more insight into the empirical hardness of instances which were lacking in other features.

Recently, Liu *et al.* [38] applied the ISA to the 2DBPP to compare MIP models. They considered the Chen's model [10], Pisinger's model [44], Blum's model [5] and Castro's model [9]. In testing these models against each other, time was the performance metric, and the models were compared in terms of the time it takes to solve a given instance. It was shown that

Chen's model was the best performing model with a 48.3%, while Pisinger's model performed the worst at 19.5%. Chen's model showed the highest precision, and was the preferred model in regions were more than one model perform good. Generating new instances improved the results, with each model's percentage of good performance improving by approximately 10%. The interesting observation was that the precision of Blums model was now the highest, meaning that in regions where more than one algorithm performs good, Blum's model is now the preferred model. The ISA was insightful in comparing the models and gave algorithm footprints which showed regions where even the strongest performing model on average was struggling. Which further emphasizes that an algorithm can perform good in a given set of instances but it does not necessarily mean that it will perform good on all instances. This paper also showed that there is a huge benefit in having a lot of instance spanning the instance space as that adds a lot of insight.

3.4 Focus of this thesis

In logistics environments, BPPs are regularly solved to support tactical capacity planning, as well as to optimize transportation and storage operations [12]. Consequently, when optimization techniques are applied in such settings to offer decision support, it is crucial that these methods are capable of swiftly generating high-quality solutions for the given BPPs.

This thesis employs the ISA method to investigate the relationship between problem instance features and algorithm performance for the GBPP for the first time. The study will compare several benchmark algorithms for the GBPP with newly proposed algorithms to gain insights into which regions of the instance space pose challenges for the algorithms in question. The primary objective is to identify which algorithm would be most suitable for solving a random instance, rather than simply relying on the algorithm that performs best on average. Specifically, in the context of the GBPP, if an instance is presented with items that need to be packed, the goal is to determine which algorithm would be most effective for that particular instance, as opposed to defaulting to the algorithm that typically performs best on average but may not be well-suited for that specific case.

This approach could prove highly beneficial for companies in the logistics industry, potentially leading to improved profit margins. By selecting the most appropriate algorithm for each instance, rather than using the best-performing algorithm on average, companies can avoid situations where the optimal algorithm for the overall problem performs poorly on certain instances. Thus, when the best algorithm on average yields suboptimal results, it would be replaced with a more appropriate alternative. The MATILDA platform will be utilized to conduct this analysis, providing the necessary tools to test and compare the algorithms within this framework.

Part II
Methodology

CHAPTER 4

IMPROVED HEURISTICS FOR THE GBPP

As outlined in Section 1.3, one of the primary objectives of this study is to propose improved algorithms for solving the GBPP. Six algorithms are proposed in this work, namely the HS1m, the nGreedy, the GRASPm, the LNSm, the GRASP_LNS, and the LNS_Local, which are inspired from existing heuristics in the literature. A detailed description of each algorithm is provided in this chapter, followed by a corresponding pseudo-code that outlines the procedure, along with a worked example to demonstrate its application.

The example utilized is based on a randomly selected data instance from benchmark instances by Baldi *et al.* [2], denoted by $\mathcal{I}$, containing 25 non-compulsory items [1], which was employed for testing all six algorithms. Regarding the bins in this instance, there are five distinct bin types with capacities of 60, 80, 100, 120, and 150. The bins are assigned costs equivalent to their respective capacities. In total, 58 bins are available, consisting solely of the five bin types. Specifically, 18 bins have a capacity of 60, followed by 13 bins with a capacity of 80, 11 bins with a capacity of 100, 9 bins with a capacity of 120, and 7 bins with a capacity of 150. The volumes and profits associated with each item are detailed in Table 4.1.

Items (I_i)	1	2	3	4	5	6	7	8	9	10	11
Volumes (v_i)	85	75	39	5	30	76	85	27	63	41	96
Profits (p_i)	172	290	118	6	47	58	76	26	70	116	247
	12	13	14	15	16	17	18	19	20	21	22
	29	32	2	30	26	7	37	56	26	13	94
	47	78	4	89	45	28	118	223	38	10	346
	23	24	25								
	24	5	23								
	46	4	48								

Table 4.1: Volumes and profits of items from the data instance $\mathcal{I}$, used as an example to showcase the working all algorithms considered in this thesis, in the order in which they appear in $\mathcal{I}$.

[1] It is easier to visualize differences in packing using only non-compulsory items given the low number of items

4.1 The HS1m algorithm

The HS1m algorithm, shown in Algorithm 4.1, is an adaptation of the HS1 algorithm presented in the work of Crainic *et al.* [13], originally designed to solve the Variable Size Bin Packing Problem with Assignment Cost problem. This algorithm has been modified to better suit the GBPP, given the differences in problem settings. The core concept behind the algorithm is to assign each item to the most suitable bin in which it can be packed, provided that such a bin has already been selected. If no suitable bin has been selected yet, the algorithm will identify the most appropriate bin for the item, along with a corresponding subset of items that can be packed in the same bin.

The algorithm begins by initializing two sets: the set of selected bins and the set of assigned items, both of which are initially empty. It then processes a list of items and bins, sequentially considering the unassigned items to determine the most appropriate bin in which each item can be packed. For each item, the algorithm first checks whether there is an already selected bin that can accommodate the item with the smallest cost contribution. In the context of this problem, the notion of "least cost contribution" has been adapted to focus on maximizing the remaining available space within a bin, thereby creating room for additional items. This strategy is designed to minimize the number of new bins opened, as it aims to efficiently pack items into already used bins. The "cost contribution" is quantified as the ratio of the residual space in the bin after packing the item to the cost of the bin, applied across all already selected bins.

A new bin will only be selected if it is deemed cost-efficient to do so. The selection of a new bin follows the guidelines of Algorithm 4.2, which recommends choosing a bin and packing an item into it if the overall cost contribution is low, ensuring that multiple items can be packed into the bin from the remaining unassigned items. Finally, Algorithm 4.3 explores potential swaps between unassigned and assigned items if such exchanges are deemed beneficial. An illustration of the packing process using this algorithm, applied to the instance $\mathcal{I}$ presented in Table 4.1, is shown in Figure 4.1 and Example 1.

Example 1 *Using the list of items as they appear in the instance, and a list of bins ordered by increasing capacities from bin 1 to bin 58, the packing process proceeds as follows: I_1 is the first to be considered and is packed into the first bin in which it could fit (bin 32), which has a capacity of 100. The residual volume is then updated accordingly. Next, I_2 is packed into bin 19 as its profit exceeded the bin cost. A new bin (bin 1) is opened for I_3, as it is deemed profitable to do so. I_4 is packed into bin 1, which was previously opened and is still profitable. I_5 is packed into bin 2. I_6 was not packed as there are no items that would fill the residual space in the bin and result in a profit.*

The next item I_7 *is packed into bin 33. I_8 remained unpacked as the item with the least contribution to fill the residual space does not result in a profit. Bin 20 is opened for I_9. I_{10} is packed into bin 3 as the profit exceeds the bin cost. I_{11} could be packed into bin 34, with the residual space not enough to pack another item as it was profitable by itself. I_{12} had a similar issue as I_8 and was not packed. I_{13} was packed into a new bin (bin 4) as it was profitable to do so. I_{14} was packed into bin 2 as it does not use a lot of the residual space. This was followed by I_{15}, for which a new bin (bin 5) was opened and profitable. I_{16} and I_{17} were not*

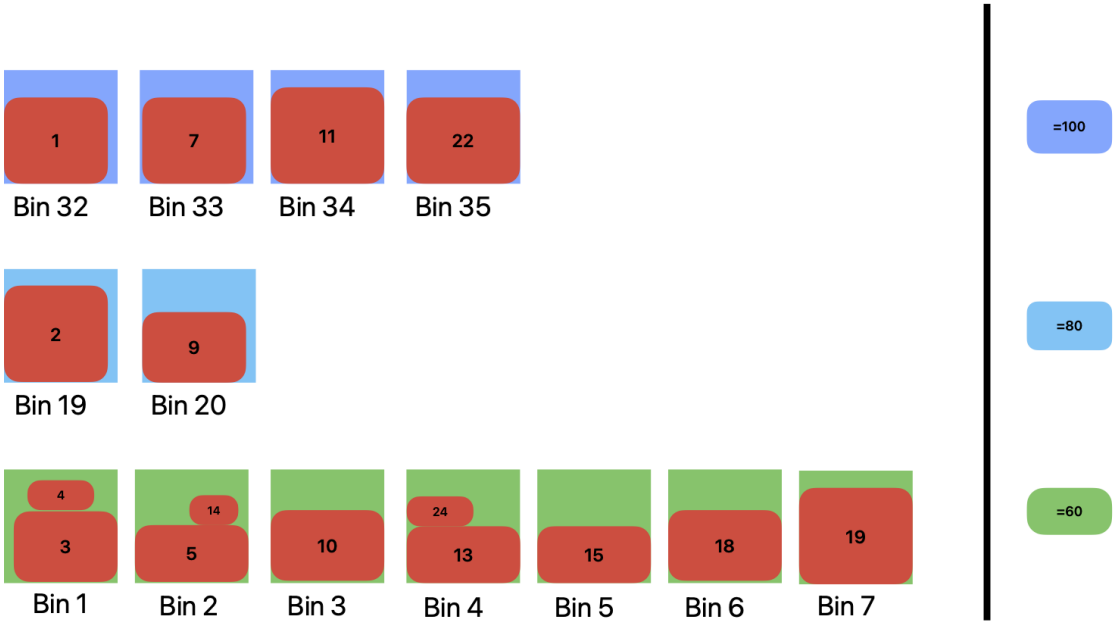


Figure 4.1: Results obtained when packing items presented in the instance $\mathcal{I}$ using the HS1m algorithm. The items are presented in red, with the numbers inside indicating their respective item number. The algorithm utilizes three bin types, with capacities 60, 80 and 100. The resulting net cost is -1024, with items $\{I_6,\ I_8,\ I_{12},\ I_{16},\ I_{17},\ I_{20},\ I_{21},\ I_{23},\ I_{25}\}$ not packed.

packed as there was another item with a lower cost contribution for used bins yet to be packed (I_{24}). Bin 6 was opened for I_{18}, and bin 7 for I_{19} as it was profitable for both bins. I_{20} and I_{21} were not packed as it was not profitable to open a new bin for them, and they could not be packed in used bins as an item with a lower contribution cost was still available. Bin 35 was opened for I_{22} as its profit exceeded the bin cost. I_{23} could also fit in a used bin, but it did not have the least cost contribution, and was skipped as a result. I_{24} was packed into bin 4 as it could fit and had the lowest cost contribution. I_{25} was not packed as there was another item with a lower cost contribution unpacked (I_{17}).

Packing using this algorithm resulted in the following result: $S_{final} = \{bin1 : I_3,\ I_4;\ bin2 : I_5,\ I_{14};\ bin3 : I_{10};\ bin4 : I_{13},\ I_{24};\ bin5 : I_{15};\ bin6 : I_{18};\ bin7 : I_{19};\ bin19 : I_2;\ bin20 : I_9;\ bin32 : I_1;\ bin33 : I_7;\ bin34 : I_{11};\ bin35 : I_{22}\}$. *The items not packed were:* $\{I_6,\ I_8,\ I_{12},\ I_{16},\ I_{17},\ I_{20},\ I_{21},\ I_{23},\ I_{25}\}$

Example 1 and Figure 4.1 demonstrate the concept of "least cost contribution" present in the HS1m algorithm. Indeed, some items are packed into bins where they may appear to have alternative placement options, especially when considering the potential to pack them into previously used bins. The bin in which an item is packed is the one that yields the highest profit after the item is placed, provided that the item fits. This ensures that the item contributes the least to the overall cost of that bin. This strategy is particularly advantageous when the item-to-bin assignment cost varies between bins. However, in the context of the GBPP, assignment costs were not included. As a result, the algorithm achieved a net cost of -1024, with 16 out of the 25 items successfully packed and only 13 out of the 58 bins utilized.

Algorithm 4.1 The HS1m algorithm

Input: SBL, SIL

Output: A feasible packing of items into the bins

```
S = ∅

for item i in SIL do
    while S is not empty do
        calculate the cost contribution for packing i into every bin in S
        if there are bins in S where item i can fit then
            fit the item in the first bin b
            update the space left in the bin
        else
            if item in question is compulsory then
                get the first bin b from SBL where the item can fit
                fit the item and update the space left add the bin to S
            else
                get new itemlist SIL_i get the first bin b in which PROFITABLE(i, b) return
                 TRUE
                fit the item and update the space left
                add the bin to S
```

Algorithm 4.2 PROFITABLE procedure for new bin selection for an item i

Input: SIL_i, b

Output: TRUE or FALSE

```
SIL_i : new item list of unassigned items
Load item i into the first bin in b in which it fits

for item i* in SIL_i do
    Calculate cost contribution
    Sort unassigned items by increasing order of cost contribution
    if i* can fit into b then
        fit i* into b
        update the bin profit to P_b = P_i + P_i*
    if P_b > C_b then
        return TRUE
    else
        FALSE
```

Algorithm 4.3 POST-OPTIMIZATION procedure

Input: SIL_i, b

Output: TRUE or FALSE

```
for item i* in SIL_i do
    if i* can swap with i then
        if PROFITABLE(i*, b) then
            swap items
```

4.2 The nGreedy algorithm

The nGreedy algorithm, provided in Algorithm 4.4, operates similarly to the U-FFD algorithm used by Baldi *et al.* [2] in the original GBPP literature, but with less complexity. It attempts to open new bins as long as there are items left to pack. Specifically, if the sum of the volumes of the unassigned items exceeds the capacity of a bin, the algorithm assumes it is beneficial to open a new bin and packs it to capacity, provided the item in question can fit. The approach is based on making a locally optimal decision with the expectation that these choices will ultimately lead to a globally optimal solution, as described in [61]. In contrast, the U-FFD algorithm takes profitability into account before deciding to open a new bin. This algorithm does not involve exploring the neighbourhood of the current solution.

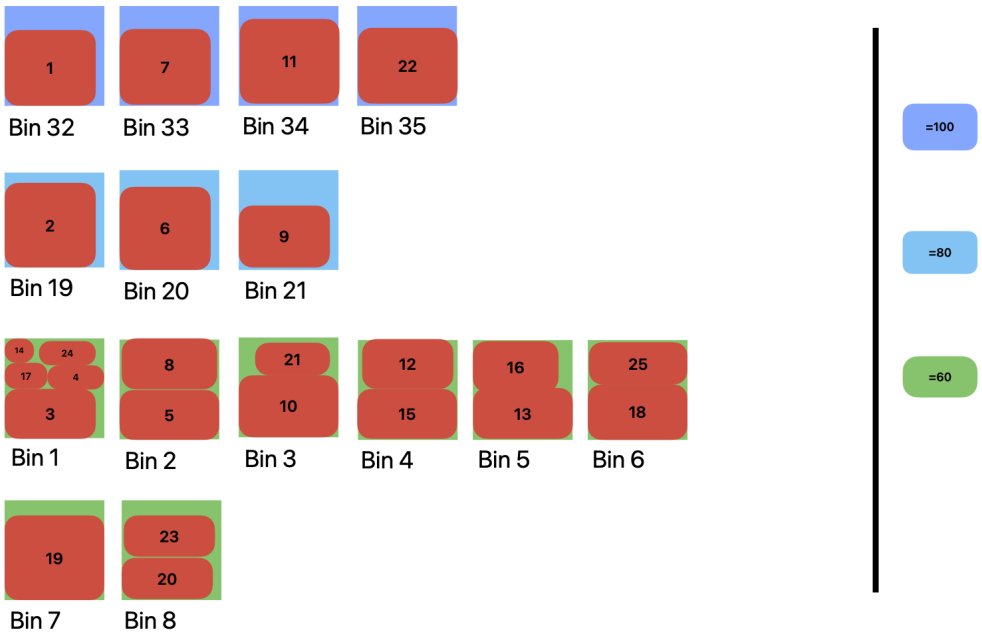


Figure 4.2: Results obtained when packing items presented in the instance $\mathcal{I}$ using the nGreedy algorithm. The items are presented in red, with the numbers inside indicating their respective item number. The algorithm utilizes three bin types, with capacities 60, 80 and 100. The resulting net cost is -1230, with all the items packed.

Example 2 *By sorting the items in the order in which they appear on the data instance and ordering the bins by increasing order from bin 1 to bin 58, items are packed in the order they appear starting from the smallest bins. Considering there are no used bins yet, the algorithm opens a new bin for I_3, which is the first item that can fit in the first bin. The item alone has profit that is greater than the bin cost, so nothing else was considered when fitting it in. I_4 was not profitable enough to open a new bin, so only the one available used bin was considered for it. The item was fitted as there was enough space. I_{14}, I_{17} and I_{24} were fitted next as they could fit in the residual space of bin 1. No other item could fit into bin 1 anymore, hence the algorithm moves to the next bin and repeats the process until no items fit. Subsequently, I_5 and I_8 were fitted into bin 2.*

Bin 3 was then considered for I_{10} and I_{21}. I_{12} and I_{15} were fitted into bin 4. The next bin (5) was opened, in which I_{13} and I_{16} were packed. I_{18} and I_{25} were fitted into bin 6, which was filled to capacity. The next item I_{19} was fitted into bin 7. Bin 8 was opened for I_{20} and

I_{23}. At this point, no other item could fit into bins of capacity '60'. Therefore the algorithm considers bins of capacity '80', which start from bin 19. This bin was opened and I_2 was packed into it. I_6 was packed in bin 20, leaving very little residual that no other item could fit. The next item I_9 was packed into bin 21. After this, no other item could fit into bins of capacity '80', therefore, moving over to bins of capacity '100'. Bin 32 was opened for I_1. I_7 was packed into bin 33. The next bin (34) was opened for I_{11}. Finally, the last bin (35) was opened for I_{22}. This process of fitting an item to a bin, and fitting other items in the residual space until there is no item that can fit was repeated until all items were packed.

The packing of items using this algorithm resulted in the following solution: $S_{final} = \{bin1 : I_3,\ I_4,\ I_{14},\ I_{17},\ I_{24};\ bin2 : I_5,\ I_8;\ bin3 : I_{10},\ I_{21};\ bin4 : I_{12},\ I_{15};\ bin5 : I_{13},\ I_{16};\ bin6 : I_{18},\ I_{25};\ bin7 : I_{19};\ bin8 : I_{20},\ I_{23};\ bin19 : I_2;\ bin20 : I_6;\ bin21 : I_9;\ bin32 : I_1;\ bin33 : I_7;\ bin34 : I_{11};\ bin35 : I_{22}\}$.

Figure 4.2 shows the results from packing items into bins using the this algorithm. A key distinction between the nGreedy algorithm and the others is that all items were successfully packed in the example. This outcome is attributed to the algorithm's lack of hesitation when opening a new bin. As long as a bin can potentially be filled, it is opened. The decision to open a new bin is made based on the assumption that, at the time of opening, it is the optimal choice, even though it could become costly if the bin is not filled to capacity. The algorithm achieved a net cost of -1230, which, in this instance, outperformed the result achieved by the HS1m algorithm. In this example, all 25 items were packed, and 16 out of the 58 available bins were utilized.

Algorithm 4.4 The nGreedy algorithm

Input: SBL, SIL, total volume of items
Output: A feasible packing of items into the bins

```
1  for item in SIL do
2      for bin in SBL do
3          if residual bin space > item volume then
4              if bin has not been opened yet, and there are enough items to pack then
5                  Open the bin
6                  Assume it will be packed to capacity
7                  Update residual space in bin
8              else
9                  pack the item
10                 update residual space in bin
11             Break
```

4.3 The GRASPm algorithm

The GRASPm algorithm, shown in Algorithm 4.5, constructs a packing solution by employing a local search technique to iteratively improve the current solution. This is based on the GASP used by Baldi *et al.* [3] to solve the GBPPI. The core principle behind GRASPm is that if a local search results in an improved solution, the current solution is updated to this new configuration, and the search continues until no further improvements can be made or

the maximum number of iterations is reached. The algorithm begins with multiple solutions derived from a variant of the U-FFD algorithm, each using different sorting strategies for items and bins. Consequently, the initial solutions for GRASPm are expected to vary, as the sorting strategy significantly influences how the U-FFD algorithm packs the items. Additionally, by combining these diverse starting points or current solutions, the GRASPm algorithm performs a local search to generate neighbouring solutions, exploring potential substitutions for the bins used in an effort to enhance the solution. The algorithm stops when no further improvements are possible, or the maximum number of iterations is reached.

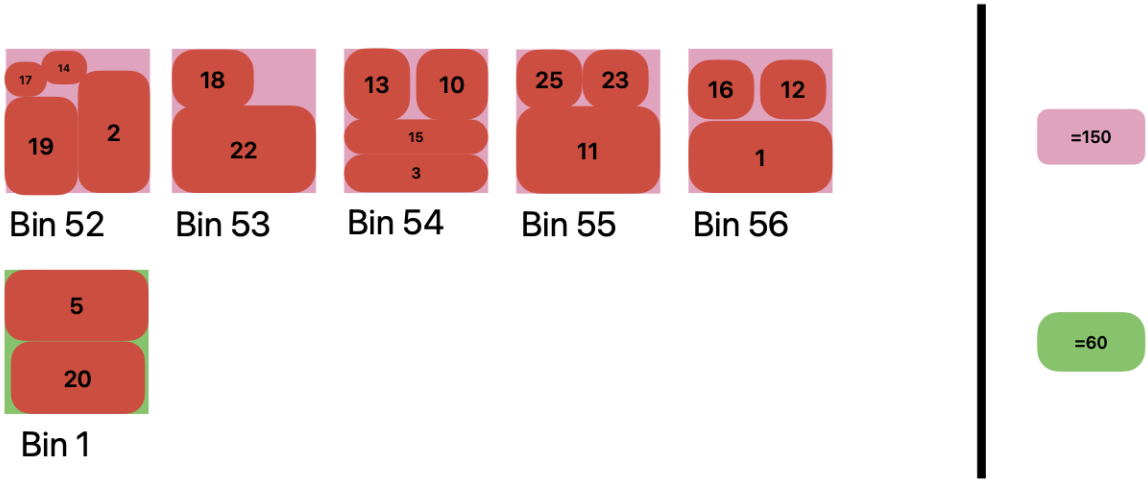


Figure 4.3: Results obtained when packing items presented in the instance $\mathcal{I}$ using the GRASPm algorithm. The items are presented in red, with the numbers inside indicating their respective item number. The algorithm utilizes two bin types, with capacities 60 and 150. The resulting net cost is -1230, with items $\{I_4,\ I_6,\ I_7,\ I_8,\ I_9,\ I_{21},\ I_{24}\}$ not packed.

Example 3 *GRASPm takes multiple solutions from a variant of the first fit algorithm, obtained using different sorting strategies, and from the nGreedy algorithm. The first solution (S_1) from the first fit at which items were packed by non-increasing order of the ratio of profit and volume, with the bins ordered by non-increasing order of volume.* $S_1 = \{bin52 : I_{17},\ I_{19},\ I_2,\ I_{14};\ bin53 : I_{22},\ I_{18};\ bin54 : I_3,\ I_{15},\ I_{10},\ I_{13};\ bin55 : I_{11},\ I_{25},\ I_{23};\ bin56 : I_1,\ I_{16},\ I_{12};\ bin43 : I_{20};\ bin32 : I_5\}$. *For the second solution (S_2) from the first fit, items were packed in non-increasing order of profits, and bins were sorted by increasing order of cost.* $S_2 = \{bin1 : I_{22};\ bin2 : I_2;\ bin3 : I_{11};\ bin4 : I_{19},\ I_{14};\ bin5 : I_1;\ bin6 : I_3,\ I_{17},\ I_{21};\ bin7 : I_{18},\ I_{25};\ bin8 : I_{10},\ I_4,\ I_{24};\ bin9 : I_{15},\ I_5;\ bin10 : I_{13},\ I_{23};\ bin11 : I_7;\ bin12 : I_9;\ bin13 : I_6;\ bin14 : I_{12},\ I_{16};\ bin15 : I_{20},\ I_8\}$.

The next solution used was obtained by sorting the items in non-increasing order of the volumes and the bins by increasing order of cost. $S_3 = \{bin1 : I_{11};\ bin2 : I_{22};\ bin3 : I_1;\ bin4 : I_7;\ bin5 : I_6;\ bin6 : I_2;\ bin7 : I_9;\ bin8 : I_{19},\ I_{14};\ bin9 : I_{10},\ I_{21},\ I_4;\ bin10 : I_3,\ I_{17},\ I_{24};\ bin11 : I_{18},\ I_{25};\ bin12 : I_{13},\ I_8;\ bin13 : I_5,\ I_{15};\ bin14 : I_{12},\ I_{16};\ bin15 : I_{20},\ I_{23}\}$. S_4 *resulted after items were ordered non-increasing profits and bins ordered by non-increasing volumes.* $S_4 = \{bin52 : I_{22},\ I_{19};\ bin53 : I_2,\ I_3,\ I_{15};\ bin54 : I_{11},\ I_{18};\ bin55 : I_1,\ I_{10};\ bin56 : I_9;\ bin44 : I_{13},\ I_7;\ bin19 : I_6\}$.

Ordering items and bins by non-increasing orders of volumes, and applying the first fit resulted

in $S_5 = \{bin53 : I_{22};\ bin55 : I_7;\ bin57 : I_2;\ bin44 : I_{11};\ bin46 : I_6;\ bin33 : I_1;\ bin1 : I_9\}$. *The next first fit solution resulted after ordering the items by non-increasing order of the ratio of profit and volume, and the bins by increasing order of cost.* $S_6 = \{bin1 : I_{17},\ I_{18},\ I_{14},\ I_4,\ I_{24};\ bin2 : I_{19};\ bin3 : I_2;\ bin4 : I_{22};\ bin5 : I_3,\ I_{21};\ bin6 : I_{15},\ I_{25};\ bin7 : I_{10};\ bin8 : I_{11};\ bin9 : I_{13},\ I_{23};\ bin10 : I_1;\ bin11 : I_{16},\ I_{12};\ bin12 : I_5,\ I_{20};\ bin13 : I_9;\ bin14 : I_8;\ bin15 : I_7\}$. *And finally, the solution obtained from the nGreedy in 4.2.*

Although some of the solutions were deemed infeasible (e.g., S_5), they were still utilized as starting points in the GRASPm algorithm, with the goal of attempting to improve upon them. Using these seven distinct solutions as initial configurations, the algorithm searches for unused bins that could replace the used bins at a lower cost. It ultimately reached the solution $S_{final} = \{bin52 : I_{17},\ I_{19},\ I_2,\ I_{14};\ bin53 : I_{22},\ I_{18};\ bin54 : I_3,\ I_{15},\ I_{10},\ I_{13};\ bin55 : I_{11},\ I_{25},\ I_{23};\ bin56 : I_1,\ I_{16},\ I_{12};\ bin1 : I_{20},\ I_5\}$, depicted in Figure 4.3. In this final solution, a net cost of -1230 was achieved, which is comparable to the result obtained by the nGreedy algorithm. However, the items are packed differently in this case, reflecting the algorithm's optimization approach and the impact of the various starting points on the resulting solution.

Algorithm 4.5 The GRASPm algorithm

Input: SBL, SIL, total volume of items, max iterations
Output: A feasible packing of items into the bins

```
1  Solutions = [ ]
2  Append multiple starting points to Solution
3  Solutions = list of multiple current solutions (current_sol)

4  for current_sol in Solution do
5      current_obj = current_sol_objective
6      for x in range max_iterations do
7          generate neighbours to current_sol
8          if there are no neighbours then
9              Break
10         else
11             if best_neighbour_obj < current_obj then
12                 Update solution and objective
13             else
14                 Break
```

Algorithm 4.6 GENERATE_NEIGHBOURHOOD

Input: Solution, SIL, SBL

Output: neighbours to the solution

```
1 get the bins not used
2 get the used bins

3 for bin in bins used do
4     for bin1 in bins not used do
5         if items in bin can fit in bin 1 then
6             if bin1.cost < bin.cost then
7                 Swap the bins
8         else
9             break
```

4.4 The LNSm algorithm

The LNSm algorithm, provided in Algorithm 4.7, operates similarly to GRASPm in that it begins with multiple starting points and attempts to improve these solutions until no further improvement can be made or the maximum number of iterations is reached. However, as suggested by its name, the LNSm algorithm employs a large neighbourhood search (LNS). This approach uses destroy and repair methods to generate new solutions. This algorithm was inspired by the Adaptive Large Neighbourhood Search (ALNS), which was used by Rodrigues *et al.* [48] to solve the GBPPIC. However, unlike the ALNS, the LNSm algorithm utilizes only one destroy method and one repair method.

The principle behind the destroy and repair methods is as follows: starting with a current solution, the algorithm "destroys" part of it by removing a portion of the items, while ensuring that the compulsory items, which must be packed, remain unaffected. The solution is then "repaired" by finding other unassigned items that can be packed into the available spaces left by the destroyed items. The goal of the repair method is to improve the current solution, while maintaining feasibility.

In the process of generating a large neighbourhood, as outlined in Algorithm 4.8, the destroy method removes all non-compulsory items, and the repair method seeks to fill the resulting empty spaces with unassigned items. If the new solution is better than the current one, the solution is updated. If not, the current solution is retained, and the algorithm proceeds to the next iteration. The best solution is either the final solution that cannot be improved further or the last feasible solution found before reaching the maximum number of iterations.

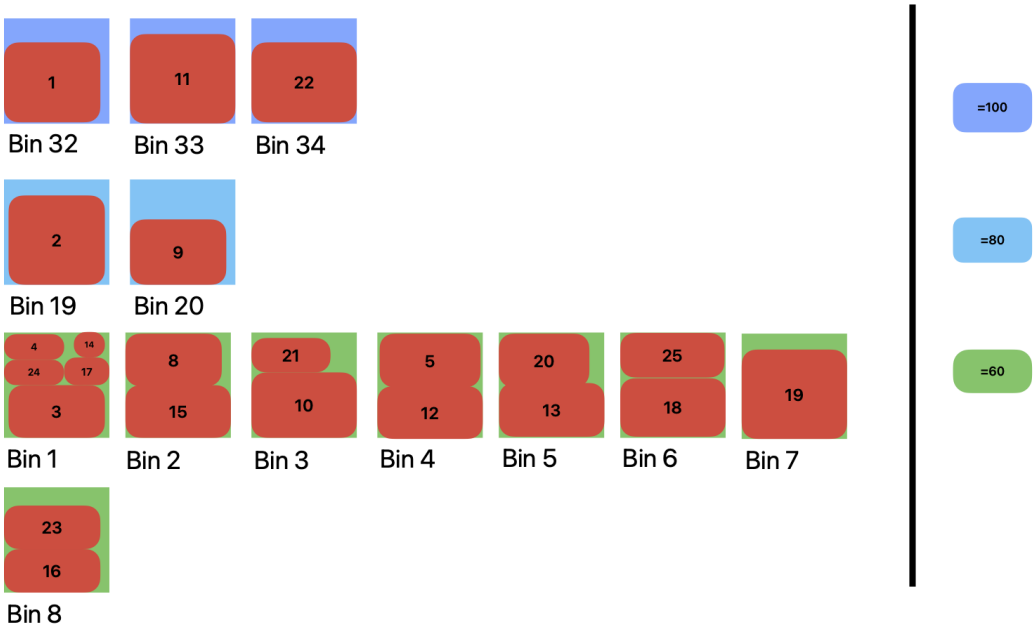


Figure 4.4: Results obtained when packing items presented in the instance $\mathcal{I}$ using the LNSm algorithm. The items are presented in red, with the numbers inside indicating their respective item number. The algorithm utilizes three bin types, with capacities 60, 80 and 100. The resulting net cost is -1276, with items $\{I_6,\ I_7\}$ not packed.

Algorithm 4.7 The LNSm algorithm

Input: SBL, SIL, total volume of items, max iterations
Output: A feasible packing of items into the bins

```
Solutions = [ ]
Append multiple starting points to Solution
Solutions = list of multiple current solutions (current_sol)

for current_sol in Solution do
    current_obj = current_sol_objective
    for x in range max_iterations do
        GENERATE_LARGE_NEIGHBOURHOOD 4.8 to current_sol
        if there are no neighbours then
            Break
        else
            if best_neighbour_obj < current_obj then
                Update solution and objective
            else
                Break
```

Algorithm 4.8 GENERATE_LARGE_NEIGHBOURHOOD

Input: Solution, SIL, SBL

Output: neighbours to the solution

```
1  get the items not packed
2  get the used bins

3  for bin in bins used do
4      for item in packed items do
5          if item is compulsory then
6              leave it in
7          else
8              remove the item
9      for item in unpacked items do
10         if item fits in any of the previously used bins then
11             Pack it into the bin
12         else
13             move on to the next item
```

Example 4 *The LNSm begins with multiple solutions, similar to those used in GRASPm, which are derived from a variant of the U-FFD algorithm using different sorting strategies, as well as from the nGreedy algorithm. At each iteration, the algorithm applies the destroy method by removing all non-compulsory items, which, in this particular example, are all of the items. The algorithm then attempts to repack the remaining items into the same bins, aiming to substitute the previously packed items with those that were not packed before. This process is repeated for a maximum of 1000 iterations, or until no further improvements can be found. The results obtained by LNSm are illustrated in Figure 4.4, with a net cost of* -1276*. This represents the best net cost achieved for the given data instance. The packing of items using this algorithm resulted in the following solution:* $S_{final} = \{bin1 : I_3,\ I_4,\ I_{14},\ I_{17},\ I_{24};\ bin2 : I_{15},\ I_8;\ bin3 : I_{10},\ I_{21};\ bin4 : I_{12},\ I_5;\ bin5 : I_{13},\ I_{20};\ bin6 : I_{18},\ I_{25};\ bin7 : I_{19};\ bin8 : I_{16},\ I_{23};\ bin19 : I_2;\ bin20 : I_9;\ bin32 : I_1;\ bin33 : I_{11};\ bin34 : I_{22}\}$.

4.5 The GRASP_LNS algorithm

The GRASP_LNS algorithm, provided in Algorithm 4.9, is designed to enhance both the GRASPm and LNSm algorithms by combining their strengths. In this approach, the initial solution is the one that GRASPm has produced, as it is considered the best solution from the GRASPm process. A LNS method is then applied to this solution using the same destroy and repair methods utilized in the LNSm algorithm. The rationale behind this approach is to provide the LNSm algorithm with a solution that has already been proven to be optimal by the GRASPm algorithm. By using this proven solution as the starting point for LNSm, it is expected that the algorithm can refine the solution further or at least maintain its high quality through the search.

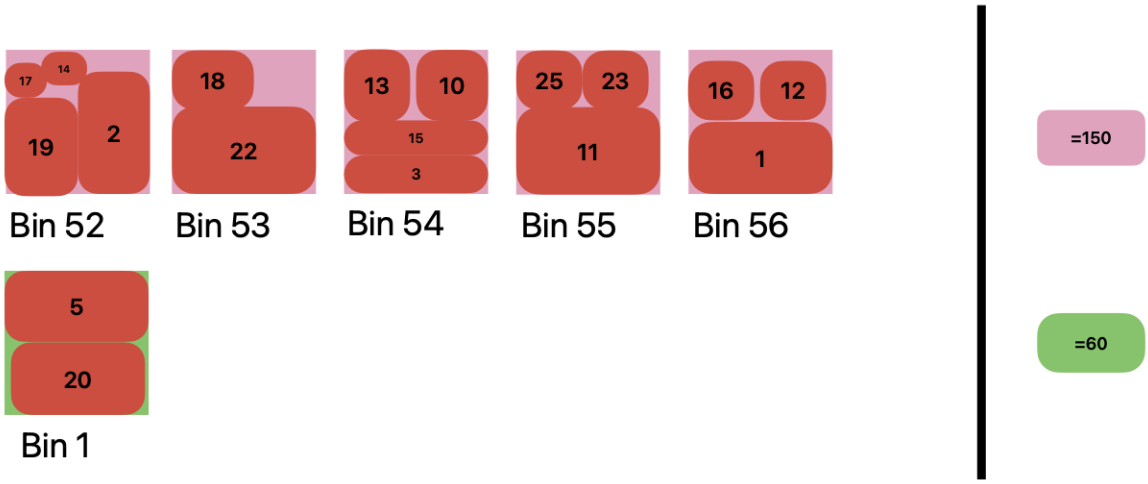


Figure 4.5: Results obtained when packing items presented in the instance $\mathcal{I}$ using the GRASP_LNS algorithm. The items are presented in red, with the numbers inside indicating their respective item number. The algorithm utilizes two bin types, with capacities 60 and 150. The resulting net cost is -1230, with items $\{I_4,\ I_6,\ I_7,\ I_8,\ I_9,\ I_{21},\ I_{24}\}$ not packed.

Algorithm 4.9 The GRASP_LNS algorithm

Input: SBL, SIL, total volume of items, max iterations
Output: A feasible packing of items into the bins
1 The procedure it the same as LNSm in algorithm 4.7, but the starting point is the solution given by the GRASPm in algorithm 4.5.

Example 5 *The GRASP_LNS algorithm begins with the solution obtained from the GRASPm algorithm. The initial solution, $S_{initial}$, consists of the following bin assignments: $\{bin52 : I_{17},\ I_{19},\ I_2,\ I_{14};\ bin53 : I_{22},\ I_{18};\ bin54 : I_3,\ I_{15},\ I_{10},\ I_{13}; bin55 : I_{11},\ I_{25},\ I_{23};\ bin56 : I_1,\ I_{16},\ I_{12};\ bin1 : I_{20},\ I_5\}$. Using this starting solution, the algorithm removes all non-compulsory items and attempts to repack them by replacing them with unpacked items. During each iteration, if the algorithm finds an improvement in the solution, it updates the current solution. The process continues for a maximum of 1000 iterations, and the final solution is the best solution obtained at the end of this process. As shown in Figure 4.5, the final solution (S_{final}) was identical to the initial solution ($S_{initial}$). The algorithm achieved a net cost of -1230, which is the same as the result produced by GRASPm, and the items were packed in a similar manner. This similarity arose because, when the number of items is small, there are limited options for repacking, meaning there are not many items that can be swapped or reassigned to different bins.*

4.6 The LNS_Local algorithm

The LNS_local algorithm, shown in Algorithm 4.10, is another hybrid approach, combining two algorithms for improved performance. In this case, the solution produced by the LNSm algorithm is used as the current best solution. Instead of applying a LNS search as in the original LNSm, this algorithm opts for a local search to generate neighbouring solutions. Similar to the GRASP_LNS approach, the idea behind the LNS local algorithm is to perform

a local search starting with a "good" initial solution. Since the solution provided by the LNSm algorithm is already considered one of the best, using it as the starting point for the local search is expected to lead to more efficient and potentially better results. The goal is to refine the current solution by exploring small changes in the solution space, leveraging the quality of the starting point to guide the search towards further improvements.

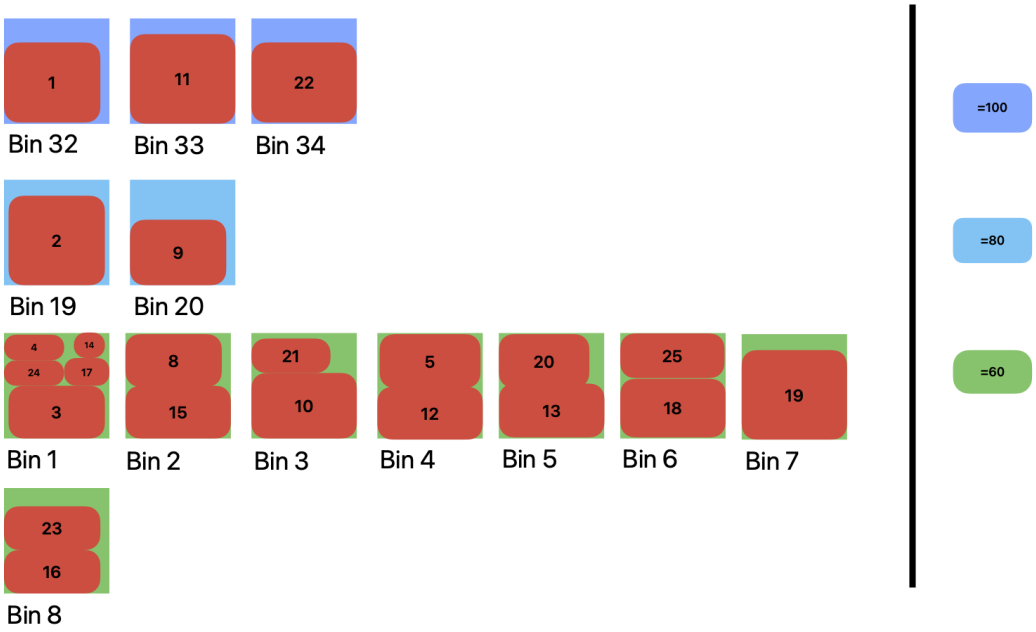


Figure 4.6: Results obtained when packing items presented in the instance $\mathcal{I}$ using the LNS_Local algorithm. The items are presented in red, with the numbers inside indicating their respective item number. The algorithm utilizes three bin types, with capacities 60, 80 and 100. The resulting net cost is -1276, with items $\{I_6,\ I_7\}$ not packed.

Example 6 *The LNS_local algorithm uses the* $S_{initial} = \{bin1 : I_3,\ I_4,\ I_{14},\ I_{17},\ I_{24};\ bin2 : I_{15},\ I_8;\ bin3 : I_{10},\ I_{21};\ bin4 : I_{12}, I_5;\ bin5 : I_{13},\ I_{20};\ bin6 : I_{18},\ I_{25};\ bin7 : I_{19};\ bin8 : I_{16},\ I_{23};\ bin19 : I_2;\ bin20 : I_9;\ bin32 : I_1;\ bin33 : I_{11};\ bin34 : I_{22}\}$*, solution provided by the LNSm algorithm as its initial solution. With this starting point, the algorithm searches for unused bins that can replace the currently used bins, aiming to improve the solution. The process is performed for a maximum of 1000 iterations. At each iteration, if a new solution is found that offers a better outcome than the current solution, the current solution is updated to reflect the improved configuration. This process continues until either the maximum number of iterations is reached or no further improvements can be found. As shown in Figure 4.6, the results returned by the LNS_Local algorithm were similar to those of LNSm for the given example. Despite the search, no unused bins were found that could replace the used bins in a way that would yield additional profit (*$S_{final} = S_{initial}$*). The net cost achieved by this algorithm was* -1276*, the same as that of LNSm, which was the best net cost obtained for the example considered.*

Algorithm 4.10 The LNS_Local algorithm

Input: SBL, SIL, total volume of items, max iterations

Output: A feasible packing of items into the bins

1 The procedure it the same as GRASPm in algorithm 4.5, but the starting point is the solution given by the LNSm in algorithm 4.7.

CHAPTER 5

APPLICATION OF ISA TO THE GBPP

The ISA framework has been applied to a variety of problems in recent years [53], including packing problems as explained in Chapter 3. However, to the best of the author's knowledge, it has not yet been applied to the GBPP. This chapter outlines the application of an ISA to the GBPP, describing the meta-data used for this purpose. Specifically, the meta-data includes the problem instances utilized, the calculated features, the algorithms considered, and the performance metrics. Once this meta-data is compiled, it is supplied to MATILDA for the implementation of the ISA framework.

5.1 Data instances

The data instances used in this project are benchmark datasets from Baldi *et al.* [2], which are publicly available and can be downloaded from their repository [16]. For the purposes of this thesis, instances with more than 500 items are excluded due to excessively long computational times for both the heuristics and the ISA algorithm. The data is summarized in Table 5.1.

Classes	Characteristics			
	no. of instances	no. of items	no. of bin types and capacities	item classification
Class 0	240	25, 50, 100, 200	3 (100, 120, 150) or 5 (60, 80, 100, 120, 150)	all compulsory
Class 1	248	25, 50, 100, 200	3 (100, 120, 150) or 5 (60, 80, 100, 120, 150)	all non-compulsory
Class 2	247	25, 50, 100, 200	3 (100, 120, 150) 5 (60, 80, 100, 120, 150)	all non-compulsory
Class gen	5000	50	5 (60, 80, 100, 120, 150)	1/3 compulsory and 2/3 non-compulsory

Table 5.1: Summary of all data instances considered for the purpose of this thesis.

Class 0 consists of instances with only compulsory items, with the number of items being 25, 50, 100, or 200 in each instance. The profit for each item is fixed at 200. The instances can either have three bin types with capacities of 100, 120, and 150, or five bin types with capacities of 60, 80, 100, 120, and 150. The cost of each bin is equal to its capacity. Classes 1

and 2 reuse the same instances as Class 0, but with all items being non-compulsory. In these classes, the profits of the items are adjusted to follow a uniform distribution. In Class 1, the profit of an item $p_i \in [U(0.5, 3)w_i]$ while in class 2, $p_i \in [U(0.5, 4)w_i]$, where w_i is the weight of the item in question.

The class gen was generated using the original instances, where both the volumes and profits of the items follow a uniform distribution. The volume of each item is randomly generated between the smallest and largest volumes from the original instances, while the profit is randomly generated between the smallest and largest profits observed in the other classes. The purpose of this class is to retain the characteristics of the original instances while providing variability for the ISA framework. All instances in this class contain 50 items due to high computational costs for upper bound calculations and the ISA. Additionally, each instance in this class includes a mixture of compulsory and non-compulsory items, with one-third of the items being compulsory and the remainder non-compulsory, all chosen randomly.

5.2 Feature selection

The selection of features is a critical step in any machine learning approach, as it directly impacts the ability of the model to represent and predict the behaviour of the problem instances effectively. In this case, the choice of features is influenced by the specific characteristics of the problem domain, the nature of the algorithms used, and the need to capture the structural properties and complexities of the instances being solved. The goal is to ensure that the features chosen not only to expose these complexities but also highlight the strengths and weaknesses of the algorithms [55].

In this specific context, the problem is one-dimensional, focusing on the volume of items. Therefore, the features are designed to revolve around this aspect, with the intention of encapsulating various facets of the instances that are relevant to the algorithm's performance. The selection of features is flexible, allowing the meta-data to be tailored in ways that provide the most insight into the problem.

No.	Feature	Description
1	n	The number of items in the instance
2	*mean*	The mean of the volumes of the items
3	*median*	The median of the volumes of the items
4	*var*	The variance of the volumes of the items
5	*max*	The volume of the largest item
6	*min*	The volume of the smallest item
7	*large*	The proportion of items that have volume $<= \frac{1}{2}$ but $>= \frac{1}{3}$
8	*medium*	The proportion of items that have volume $<= \frac{1}{3}$ but $>= \frac{1}{4}$
9	*small*	The proportion of items that have volume $<= \frac{1}{4}$
10	*tiny*	The proportion of items that have volume $<= \frac{1}{10}$
11	*mini*	The proportion of items that have volume $<= \frac{1}{20}$

Table 5.2: Description of features selected for the ISA framework used in this work.

In total, 11 features have been considered for use within the ISA framework, as detailed in Table 5.2. These features primarily describe the volumes of items in each instance. For

consistency in calculations, the volumes are normalized. Feature 1, for example, simply measures the number of items in each instance, which is a crucial factor since larger instances require more computational effort to solve effectively. Features 2 to 6 are statistical measures of item volumes, which provide insight into the distribution and variability of the items within the instance. Features 7 to 11 further characterize the volume distributions and help differentiate between instances with different volume profiles.

Although the features are designed to represent various characteristics of the instances, it is not immediately clear how these features correlate with the difficulty of an instance. Based on prior research by Schwerin and Wascher [50], it is hypothesized that instances with a mean volume of around one-third are very difficult, and should present more difficulty for the algorithms. This expectation arises from the fact that instances with larger volumes tend to be easier to solve, as the items can generally be packed into fewer bins. Conversely, instances with very small items are typically easier to solve as well because the items can be packed densely into a small number of bins [38].

To determine which features are most relevant for predicting algorithm performance, a subset of features will be chosen based on correlation analysis, ensuring that only those features that are truly predictive of performance are retained. Additionally, clustering techniques will be used to avoid redundancy and ensure that the selected features provide complementary information about the instance difficulty. All of this process is done automatically within the MATILDA platform. It is noted that this process will ultimately lead to the identification of an optimal feature set that can reliably predict the algorithm's performance on various instances.

5.3 Algorithms under consideration

A total of eight algorithms are considered for the ISA. This includes the two algorithms, U-FFD and U-BFD, by Baldi *et al.* [2] presented in Section 2.3, and the six proposed algorithms presented in Chapter 4. It is noted that the ISA framework is used to automatically predict the performance of each algorithm based on instance characteristics. This allows for a more efficient and targeted selection of algorithms, which is key to solving complex problems like the GBPP in an optimal or near-optimal manner.

5.4 Performance Metric

For an instance I and an algorithm A, The performance metric is defined as:

$$\frac{BestBound(I) - NetCost(A)}{BestBound(I)}$$

The best bound is the lower bound to the original GBPP model formulation by Baldi et al [2], which was found by dropping some constraints and treating the problem as an aggregate knapsack problem. For this thesis, the lower bound formulation was done using Gurobi optimization version 11.0.0, due to its efficiency in finding good feasible solutions within a limited computing time (60s) [28]. The formulation was found in the original GBPP paper by Baldi et al. [2]. The Net cost is the objective function value returned by all algorithms after

packing items into bins.

An algorithm can be considered to perform good on an instance if it is within a certain percentage of the best performing algorithm, 'relative', or if the misclassification rate is lower than a certain percentage, 'absolute'. For the GBPP, absolute performance will be considered.

5.5 Meta-data set-up

The meta-data set up follows the specified naming convention used in the MATILDA platform. It is organised into four types of columns. The column Instances, which contains the names of the instances. The column Sources, which represent the different classes of instances (here the class 0, class 1, class 2, and class gen). All the columns for the Features, which contain the normalized value of the calculated features of all instances. Finally, the Algorithm columns, which contain the performance metric for each algorithm with respect to each instance.

Part III
Appraisal of methodology

CHAPTER 6

EVALUATION OF THE PROPOSED HEURISTICS

The relative performance of the eight algorithms considered in this thesis was compared in terms of solution quality. The results from the six proposed improved heuristics were compared to the benchmark U-FFD AND U-BFD, and statistical tests were conducted to test whether there is any difference in the algorithms. All Algorithms were coded in Spyder IDE version 5.4.5 using Python version 3.9.14 [60], and in Rstudio version 2022.07.1 using R version 4.2.0 [49]. The algorithms were all executed on a 1.8GHz Dual-core Intel core i5 with 8GB RAM in MacOS 12.7.6 operating system.

6.1 Comparison in terms of percentage gap

The comparison of the relative performance of the eight algorithms based on solution quality, specifically the percentage gap between the average net cost achieved by each algorithm (upper bound) and the average lower bound [1], provides a clear understanding of how each algorithm performed across different instance classes. The lower bound values were computed according to the original GBPP literature by Baldi *et al.* [2], and represent the best solution that can be achieved on the instances regardless of the algorithm used. Figure 6.1 illustrates the average percentage gap for all eight algorithms with respect to the data instances from the literature.

Four of the improved heuristics (GRASPm, LNSm, GRASP_LNS, and LNS_local) achieved a reduced percentage gap to the lower bound similar to or better than the benchmark algorithms (U-FFD and U-BFD). The nGreedy algorithm performed close to the benchmarks, indicating that it is a strong competitor. The HS1m algorithm performed worse than the other algorithms, as indicated in Figure 6.1. Specifically, the GRASPm and LNSm algorithms seem to perform consistently well across instances, showing a significant reduction in the gap to the lower bound.

The GRASP_LNS and the LNS_Local, both being improvements over GRASPm and LNSm, also perform well, with only minor differences in their effectiveness. The nGreedy algorithm

[1] $Percentage\ gap = \frac{Lower\ bound - Upper\ bound}{Lower\ Bound}$

showed good performance, though not consistently at the level of the best-performing heuristics. Finally, the HS1m algorithm appears to have the highest gap to the lower bound, indicating that the heuristic might struggle with certain types of instances.

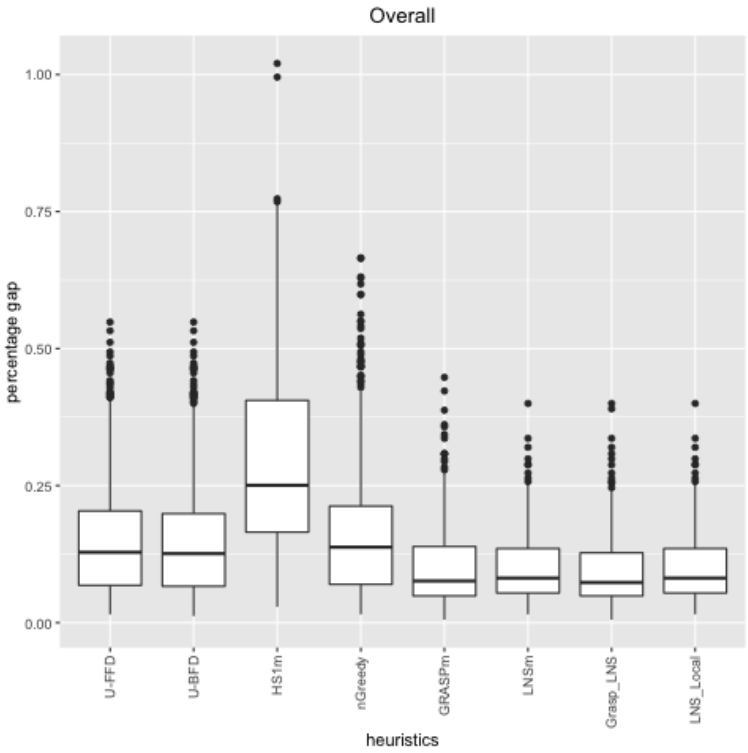


Figure 6.1: Boxplot showcasing the percentage gap achieved by each of the eight heuristics presented in Chapter 4, with respect to all data instances explained in Section 5.1.

	Algorithms	Lower bound	Upper bound	% gap
Averages	U-FFD	−8266.2	−7282.030	11.906
	U-BFD	−8266.2	−7303.524	11.646
	HS1m	−8266.2	−5834.873	29.413
	nGreedy	−8266.2	−7205.245	12.835
	GRASPm	−8266.2	−7578.461	8.32
	LNSm	−8266.2	−7564.505	8.489
	GRASP_LNS	−8266.2	−7606.910	7.976
	LNS_Local	−8266.2	−7564.505	8.489
Mean		−8266.2	−7242.507	12.384

Table 6.1: A comparison of average percentage gap from the lower bound and the net cost achieved by all algorithms with respect to all data instances.

As shown in Table 6.1, the GRASP_LNS algorithm achieved the lowest percentage gap of 7.976%, making it the best performing algorithm. The GRASPm followed closely with 8.32%. The LNSm and LNS_local algorithms achieved a percentage gap of 8.489%, indicating they performed similarly. Additionally, the U-FFD and U-BFD algorithms had significantly higher percentage gaps of 11.906% and 11.646%, respectively, showing that the proposed heuristics performed better. In general, the average gap decreased to 9.952%, showing that the proposed algorithms contribute positively to reducing the gap.

These general results corroborate with the performance of all algorithms with respect to each class of instances. A summary of the percentage gap achieved by all algorithms for each class is shown in Table 6.2. The GRASP_LNS algorithm performed consistently well across all classes, achieving the best results in Class 1 and Class 2 instances and maintaining competitive performance in Class 0. It outperformed the benchmark algorithms U-FFD and U-BFD by reducing the percentage gap from 6.854% (U-FFD) and 6.677% (U-BFD) to 6.284% with respect to class 0. The GRASPm algorithm followed closely, with slight increases in percentage gap in comparison to the GRASP_LNS. The LNSm and LNS Local showed similar performance, slightly worse than the GRASP_LNS and GRASPm but still offering significant improvements over the benchmark algorithms. Finally, the U-FFD and U-BFD algorithms showed substantially higher percentage gaps, especially in Class 1 and Class 2.

Algorithms		Classes		
		class 0	class 1	class 2
U-FFD	Lower bound	−12808.170	−4629.524	−7504.352
	Upper Bound	−11930.250	−3567.851	−6494.757
	% gap	6.854	22.933	13.453
U-BFD	Lower bound	−12808.170	−4629.524	−7504.352
	Upper Bound	−11953	−3588.923	−6515.453
	% gap	6.677	22.477	13.178
HS1m	Lower bound	−12808.170	−4629.524	−7504.352
	Upper Bound	−8467.75	−3302.31	−5819.429
	% gap	33.888	28.668	22.453
nGreedy	Lower bound	−12808.170	−4629.524	−7504.352
	Upper Bound	−11922.08	−3444.016	−6398.538
	% gap	6.918	25.608	14.736
GRASPm	Lower bound	−12808.170	−4629.524	−7504.352
	Upper Bound	−12003.25	−3994.802	−6877.239
	% gap	6.284	13.71	8.357
LNSm	Lower bound	−12808.170	−4629.524	−7504.352
	Upper Bound	−11922.08	−4013.774	−6895.526
	% gap	6.918	13.301	8.113
GRASP_LNS	Lower bound	−12808.170	−4629.524	−7504.352
	Upper Bound	−12003.25	−4041.387	−6915.121
	% gap	6.284	12.704	7.852
LNS_Local	Lower bound	−12808.170	−4629.524	−7504.352
	Upper Bound	−11922.08	−4013.774	−6895.526
	% gap	6.918	13.301	8.113

Table 6.2: A comparison of the average percentage gap from the lower bound and the net cost achieved by all algorithms with respect to different classes of data instances.

The further exploration into the percentage gap across different classes, especially by dividing the instances into groups based on the number of items, provides valuable insights. The instances were divided into four groups based on the number of items: 25 items, 50 items, 100 items, and 200 items. This segmentation allowed for a detailed comparison of how the algorithms performed as the number of items increased, as shown in Table 6.3. The improvement in the percentage gap remained consistent across all algorithms as the number

of items increased. This means that, regardless of the size of the problem (in terms of the number of items), most of the algorithms were still able to outperform the U-FFD and U-BFD benchmarks. The GRASP_LNS continued to show the best performance across all groups, regardless of the number of items in the instance. It consistently achieved the lowest percentage. The GRASPm also performed well, though slightly worse than GRASP_LNS, still showing strong results across all groupings. The LNSm and LNS_local followed the GRASPm and the GRASP_LNS closely in performance, with only minor differences in percentage gap across the different item groups. These algorithms showed robust performance. The nGreedy algorithm consistently performed close to the U-FFD and U-BFD benchmarks, especially as the number of items increased. This result was consistent with the overall percentage gap in Table 6.1, where nGreedy's performance was found to be close to that of the benchmarks.

Algorithms		Number of items			
		25	50	100	200
U-FFD	Lower bound	−2220.861	−4366.819	−8903.177	−17799.07
	Upper Bound	−1937.689	−3787.642	−7820.039	−15874.9
	% gap	12.751	13.263	12.166	11.316
U-BFD	Lower bound	−2220.861	−4366.819	−7504.352	−17799.07
	Upper Bound	−1938.611	−3796.181	−7839.541	−15842.65
	% gap	12.709	13.068	11.947	10.992
HS1m	Lower bound	−2220.861	−4366.819	−7504.352	−17799.07
	Upper Bound	−1645.961	−3154.902	−6283.967	−12409.2
	% gap	25.886	27.753	29.419	30.282
nGreedy	Lower bound	−2220.861	−4366.819	−7504.352	−17799.07
	Upper Bound	−1910.372	−3748.036	−7735.829	−15626.7
	% gap	13.981	14.17	13.112	12.205
GRASPm	Lower bound	−2220.861	−4366.819	−7504.352	−17799.07
	Upper Bound	−2023.539	−3957.492	−8134.105	−16408.08
	% gap	8.885	9.374	8.638	7.815
LNSm	Lower bound	−2220.861	−4366.819	−7504.352	−17799.07
	Upper Bound	−2017.622	−3953.922	−8122.066	−16373.14
	% gap	9.151	9.455	8.773	8.011
GRASP_LNS	Lower bound	−2220.861	−4366.819	−7504.352	−17799.07
	Upper Bound	−2040.106	−3983.016	−8167.989	−16446.03
	% gap	8.139	8.789	8.258	7.601
LNS_Local	Lower bound	−2220.861	−4366.819	−7504.352	−17799.07
	Upper Bound	−2017.622	−3953.922	−8122.066	−16373.14
	% gap	9.151	9.455	8.773	8.011

Table 6.3: A comparison of the average percentage gap from the lower bound and the net cost achieved by all algorithms with respect to the number of items on data instances.

6.2 Comparison in terms of net cost

After evaluating the percentage gap, the focus shifted to comparing the net cost achieved by the algorithms. This comparison was essential to understand the absolute performance differences between the algorithms across all data instances from the literature. Figure 6.2 illustrates the

net cost achieved by the eight algorithms. Although the figure presents the raw differences, the data does not show any striking differences between the algorithms, especially in terms of net cost. As a result, statistical testing was necessary to draw meaningful conclusions from the observed results.

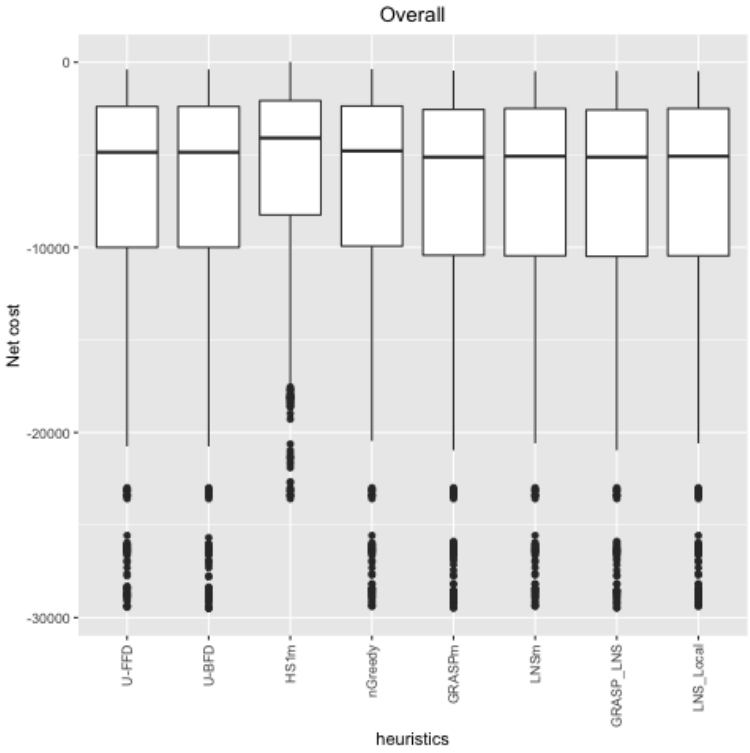


Figure 6.2: Boxplot showcasing the net cost achieved by each of the eight algorithms presented in Chapter 4, with respect to all data instances explained in Section 5.1.

The Friedman test[2] was conducted to assess whether there were significant differences in the performance of the algorithms. The null hypothesis for this test was that all the algorithms have equal mean ranks. The results of the Friedman test returned a p-value of $< 2.2e^{-16}$, which is highly significant at the 0.1% significance level. This indicates that the algorithms do indeed have significantly different performance in terms of net cost. Following the Friedman test, a Nemenyi test[3] was performed to determine the specific differences between the proposed algorithms and the benchmark heuristics (U-FFD and U-BFD). The results in Table 6.4 revealed no significant statistical difference, except for the HS1m when compared to the benchmark algorithms. The GRASP_LNS algorithm came closest to showing a significant difference with a p-value of 0.1768, suggesting that any rejection of the null hypothesis (that the U-FFD and GRASP_LNS have similar mean ranks) would be wrong 17.68% of the time. Based on this, the analysis shifts toward exploring the results for each class individually to determine if there are any significant differences between the algorithms in each class.

Figures 6.2 and Table 6.5 present the results in terms of the p-values obtained from the ad hoc test with respect to class 1 data instances. The results with respect to class 0 and class 2 instances are shown in the Appendix. According to these results, the GRASP_LNS algorithm emerged as the most significant algorithm in Class 1, showing the lowest p-value

[2] A non-parametric test used to test whether multiple algorithms differ

[3] An ad hoc test used to test which specific algorithms are significantly different after the Friedman test showed there is a difference in general

Algorithms	p-value
U-FFD and HS1m	0.00027∗ ∗ ∗
U-FFD and nGreedy	0.6813
U-FFD and GRASPm	0.2329
U-FFD and LNSm	0.2235
U-FFD and GRASP_LNS	0.1768
U-FFD and LNS_Local	0.2235

(a) Pairwise comparison of the net cost returned by the six algorithms against the U-FFD with respect to the data instances presented in Section 5.1.

Algorithms	p-value
U-BFD and HS1m	0.00022∗ ∗ ∗
U-BFD and nGreedy	0.6140
U-BFD and GRASPm	0.2690
U-BFD and LNSm	0.2573
U-BFD and GRASP_LNS	0.2047
U-BFD and LNS_Local	0.2573

(b) Pairwise comparison of the net cost returned by the six algorithms against the U-BFD with respect to the data instances presented in Section 5.1.

Table 6.4: Pairwise comparison of the net cost returned by the six algorithms against the U-FFD and U-BFD with respect to the data instances presented in Section 5.1.

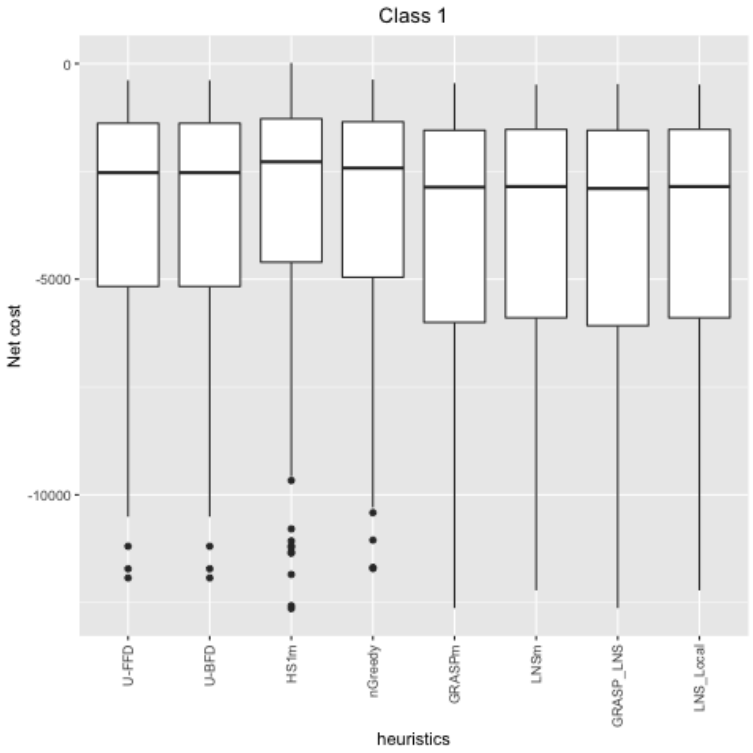


Figure 6.3: Boxplot showcasing the net cost achieved by each of the eight heuristics presented in Chapter 4, with respect to class 1 data instances explained in Section 5.1.

Algorithms	p-value (class 1)
U-FFD and HS1m	0.2129
U-FFD and nGreedy	0.5469
U-FFD and GRASPm	0.1208
U-FFD and LNSm	0.0871.
U-FFD and GRASP_LNS	0.0712.
U-FFD and LNS_Local	0.0871.

Table 6.5: Pairwise comparison of the net cost returned by the six algorithms against the U-FFD, with respect to the class 1 data instances presented in Section 5.1.

(0.0712), suggesting a strong statistical difference from the U-FFD benchmark[4]. The LNSm and LNS_Local algorithms also showed significant differences but at a less stringent level (10% significance). The GRASPm algorithm was close to showing a significant difference with U-FFD, although the p-value (0.1208) suggests that its performance may not be significantly different at the 5% significance level. In summary, these results highlight that the proposed heuristics, particularly the GRASP_LNS, are effective in improving net cost compared to the benchmark algorithms in Class 1 instances.

[4]U-FFD and U-BFD had a p-value of 0.9232, indicating that although their performance was not similar, it is very close, resulting in comparison with U-FFD only

CHAPTER 7

RESULTS FROM THE ISA ON THE GBPP

This chapter details the results obtained from applying the ISA method presented in Chapter 5 to the GBPP. The approach was executed with 735 instances sourced from literature, as described in Section 5.1. The meta-data was initially used to generate insights into the problem. In order to gain further understanding and explore new perspectives, adjustments were made to the meta-data. These adjustments aimed to improve or refine the understanding of feature impact on algorithm performance. Details and further analysis of the modified meta-data and its application are presented in the last section of this chapter.

7.1 Results of the initial instance space analysis

7.1.1 Feature selection

The 11 selected features from Section 5.2 were used to create a feature subset that best predicts good and bad algorithm performance for the GBPP. After applying feature selection techniques on MATILDA, the optimal subset of features was chosen based on their predictive ability. The chosen features were: variance (the variance of the volumes of items), min (the smallest volume of items) and mini (the proportion of items with volumes less than $\frac{1}{20}$). The projection into the 2D instance space is given below:

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} 0.1952 & -0.5036 \\ 0.2254 & 0.1577 \\ -0.5441 & -0.2089 \end{bmatrix}^T \begin{bmatrix} \text{var} \\ \text{min} \\ \text{mini} \end{bmatrix}$$

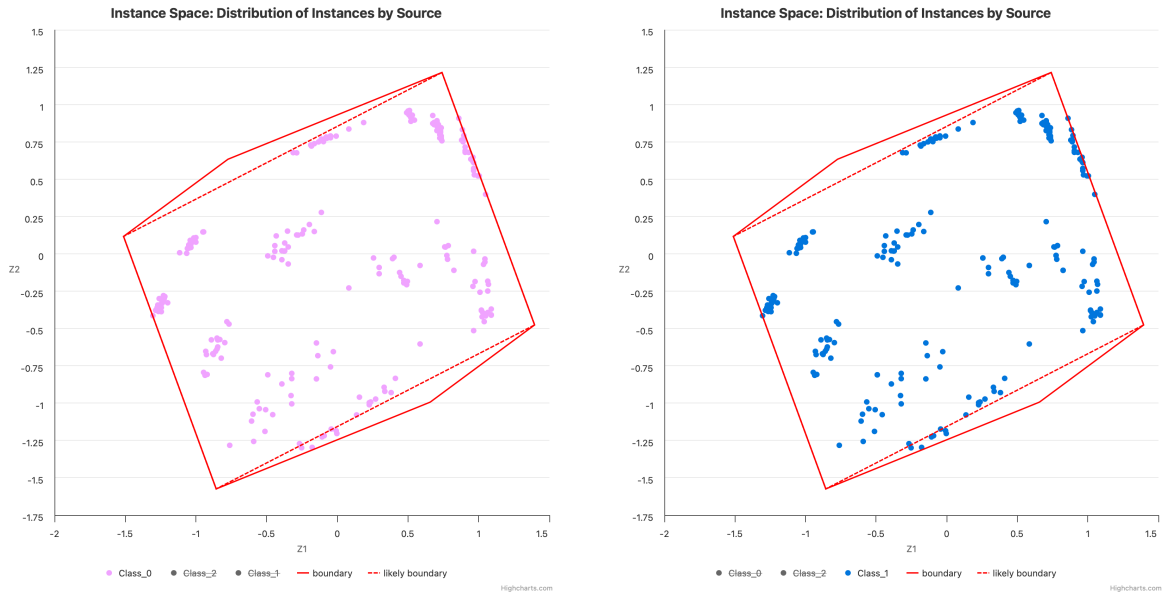
where Z_1 and Z_2 are the principal components.[1]

[1]Principal Component Analysis is used to reduce the dimensions into 2D.

7.1.2 Data visualization

To visualize and understand the relationship between the selected features and the performance of the algorithms, a 2D projection of the instance space was created. This projection would allow for a clearer understanding of how these features influence the instance difficulty, helping to differentiate between instances that are easy or hard for algorithms to solve. The 2D instance space will show how these selected features correlate with algorithm performance, allowing for insights into how each algorithm may behave in different regions of the feature space.

In Figure 7.1, the instances are effectively scattered across the instance space using the selected features. The projection helps visualize how different classes of instances are distributed and how they relate to each other in terms of algorithm performance. Class 0 instances, which only contain compulsory items with fixed profits, dominate the instance space. This is because the other classes (1 and 2) are built on top of class 0, making their distribution heavily overlap with class 0 instances. As a result, the points representing other classes are hidden underneath the class 0 points, as they correspond to one point on the instance space making the overall projection appear to show a single class.



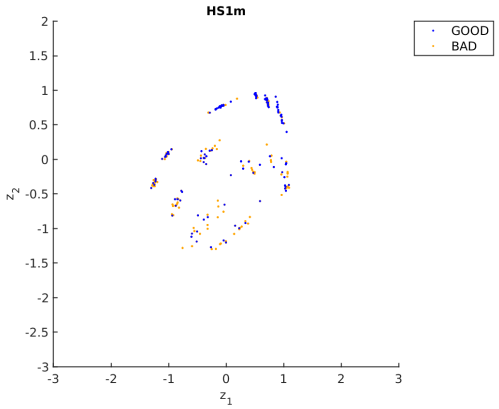
(a) A 2D scatter plot of class 0 instances from the literature in the instance space.

(b) A 2D scatter plot of class 1 instances from the literature in the instance space.

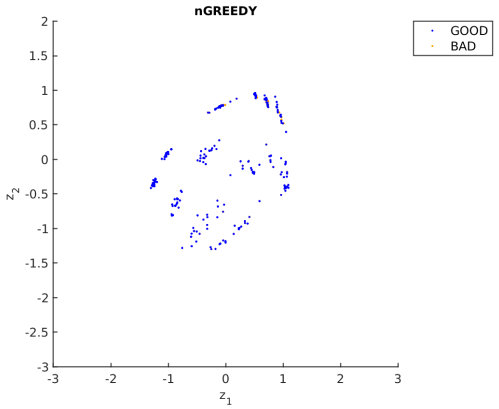
Figure 7.1: A 2D scatter plot for class 0 and class 1 instances based on the selected features of Section 7.1.5.

7.1.3 Performance distribution

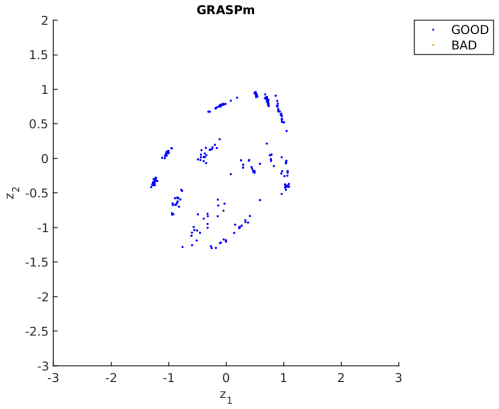
The analysis of the performance distribution in Figure 7.2 and Table 7.1 provides some valuable insights into the effectiveness of the different algorithms across the instances. As shown in Figure 7.2, instances where the algorithms perform well are highlighted in blue, while orange indicates poor performance. The algorithms tend to perform well in most instances, with the occasional exception where their performance degrades. However, the poor performance instances are scattered across the instance space and not clustered in any



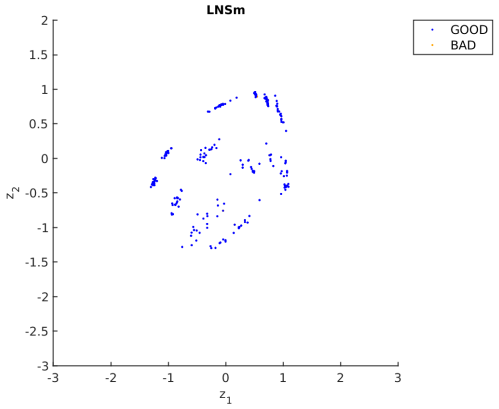
(a) Performance distribution of the HS1m algorithm over the instance space, with blue showing good performance and orange showing bad performance.



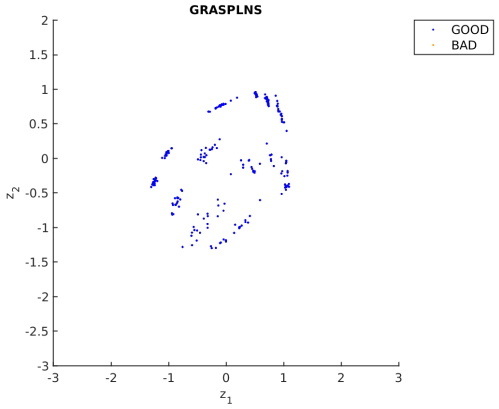
(b) Performance distribution of the nGreedy algorithm over the instance space, with blue showing good performance and orange showing bad performance.



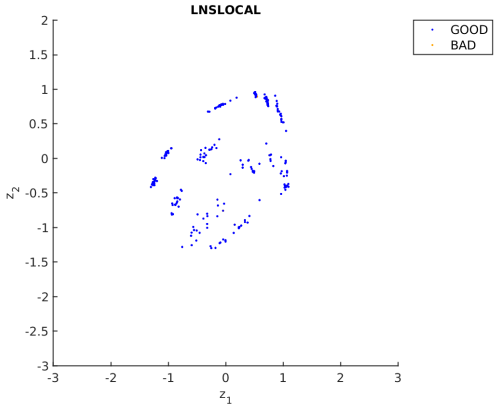
(c) Performance distribution of the GRASPm algorithm over the instance space, with blue showing good performance and orange showing bad performance.



(d) Performance distribution of the LNSm algorithm over the instance space, with blue showing good performance and orange showing bad performance.



(e) Performance distribution of the GRASP_LNS algorithm over the instance space, with blue showing good performance and orange showing bad performance.



(f) Performance distribution of the LNS_Local algorithm over the instance space, with blue showing good performance and orange showing bad performance.

Figure 7.2: The performance distribution of the six proposed algorithms over the instance space, with blue showing good performance and orange showing bad performance.

one area, suggesting that there isn't a specific instance type or class where an algorithm

consistently fails. In particular, the HS1m algorithm has the lowest percentage of good performance, with only 35.9% of instances where it performs well. The U-FFD and U-BFD algorithms perform relatively well, achieving 74% and 75.8% good performance, respectively. The nGreedy algorithm follows closely with 69.5% of instances where it performs well, which indicates it is fairly competitive with the benchmarks. The GRASPm, LNSm, GRASP_LNS, and LNS_local show excellent performance, with the percentage of good performance ranging from 89.3% to 93.5%. Among these, the GRASP_LNS algorithm performs the best.

Algorithm	Actual % of good performance
U-FFD	74%
U-BFD	75.8%
HS1m	35.9%
nGreedy	69.5%
GRASPm	89.3%
LNSm	91.8%
GRASP_LNS	93.5%
LNS_Local	91.8%

Table 7.1: Performance distribution table indicating the percentage of good and bad performance for each algorithm with respect to original data instances.

7.1.4 Algorithm footprints

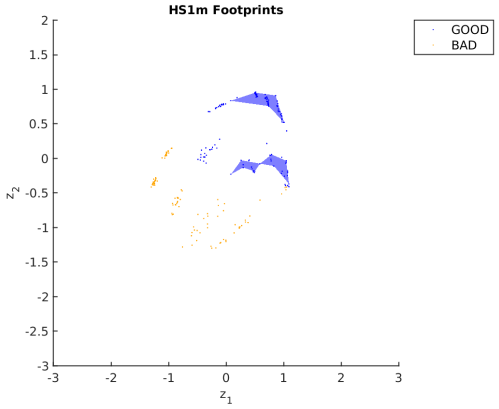
In this section, the performance of the different algorithms is visually explored within the instance space, and the SVM selector embedded in MATILDA is used to recommend the most suitable algorithm for each region. As shown in Figure 7.3, the HS1m is expected to perform well in the top-right region of the instance space. This suggests that HS1m might be better suited for specific types of instances, particularly those with certain characteristics. The nGreedy and GRASPm algorithms are expected to perform well in the bottom-left region, which might correspond to instances that are easier to solve. Like the nGreedy and GRASPm, the LNSm and LNS_Local algorithms show good performance in the bottom-left region, but their performance spans a larger area in the instance space. This suggests that they are more versatile across various instance types. The GRASP_LNS algorithm shows wide applicability, performing well almost everywhere in the instance space. This makes it the most flexible and consistently good-performing algorithm.

The result suggests that the GRASP_LNS algorithm is the best algorithm to use across all regions in the instance space. This aligns with the previous observation that the algorithm is expected to perform well almost universally, with high accuracy[2], precision[3], and recall[4] (see Table 7.2). Nonetheless, the lack of diversity in the instance space is highlighted as a limitation here, because it leads to GRASP_LNS being recommended for every region (see Figure 7.4). This could imply that the model hasn't been tested on a sufficiently diverse set of instances, and therefore, the SVM classifier has defaulted to recommending the algorithm

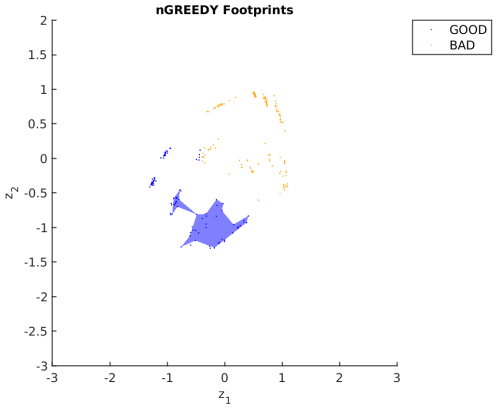
[2]Accuracy is the measure of instances that the SVM model correctly classified

[3]The precision is a measure of how precise the svm prediction is, thus, the number of true positives over true positives and false positives

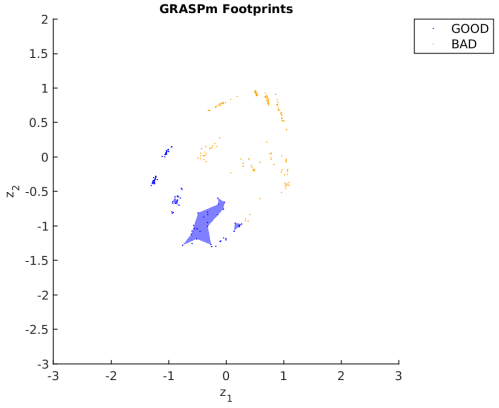
[4]Recall is the number of true positives over the number of true positives and false negatives



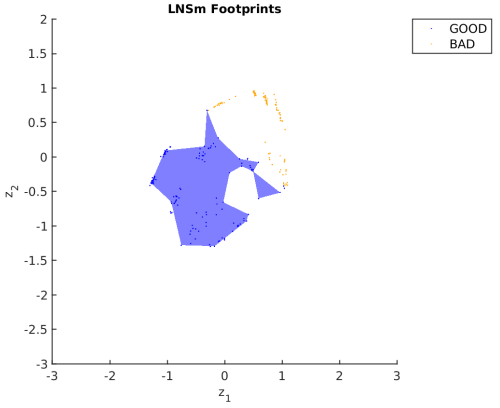
(a) Performance prediction of the HS1m algorithm over the instance space, with the region of expected strong performance highlighted in blue.



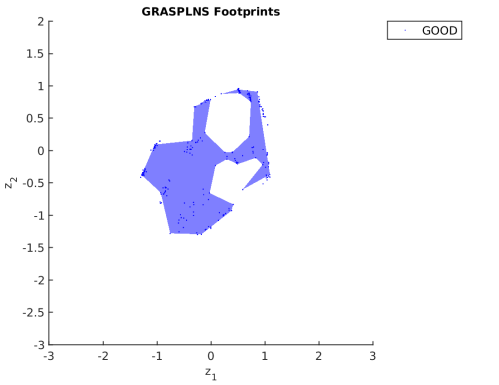
(b) Performance prediction of the nGreedy algorithm over the instance space, with the region of expected strong performance highlighted in blue.



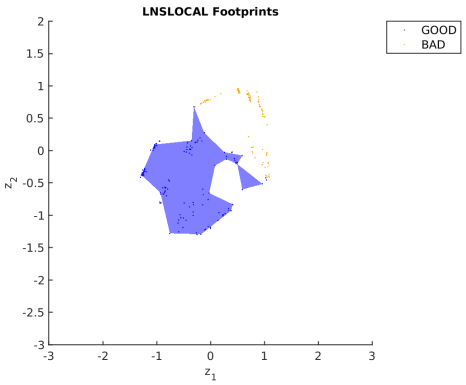
(c) Performance prediction of the GRASPm algorithm over the instance space, with the region of expected strong performance highlighted in blue.



(d) Performance prediction of the LNSm algorithm over the instance space, with the region of expected strong performance highlighted in blue.



(e) Performance prediction of the GRASP_LNS algorithm over the instance space, with the region of expected strong performance highlighted in blue.



(f) Performance prediction of the LNS_Local algorithm over the instance space, with the region of expected strong performance highlighted in blue.

Figure 7.3: The performance predictions of the six algorithms over the instance space, with the region of expected strong performance highlighted in blue.

that performs best across the board.

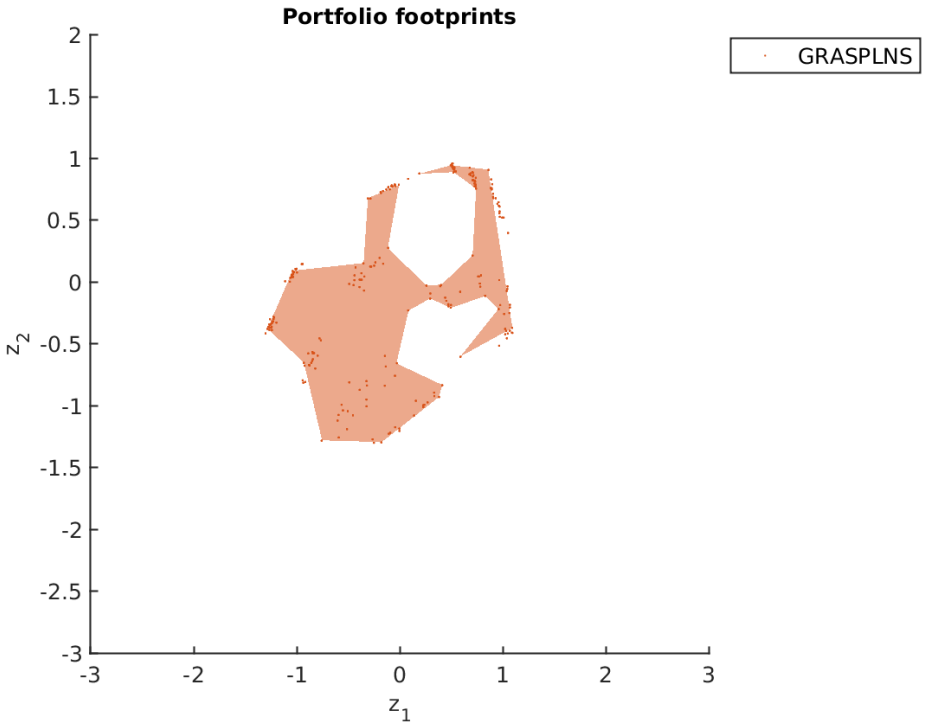


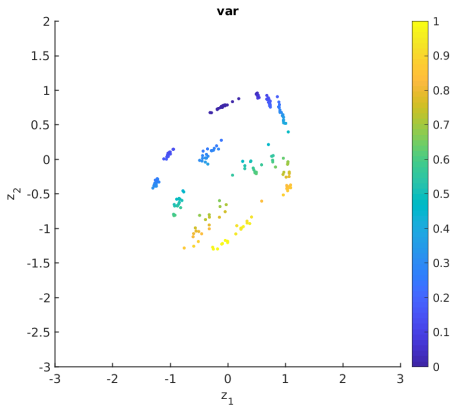
Figure 7.4: The SVM recommendation of algorithms to use for each region of the instance space.

Algorithm	Accuracy	precision	recall
U-FFD	59.3%	83.2%	56.4%
U-BFD	60.7%	76%	70.4%
HS1m	63%	49%	76.5%
nGreedy	56.3%	49%	76.5%
GRASPm	44.9%	91.4%	42.2%
LNSm	58.1%	93.4%	58.5%
GRASP_LNS	93.5%	93.5%	100%
LNS_Local	58.1%	93.4%	58.5%
Selector	N/A	93.5%	N/A

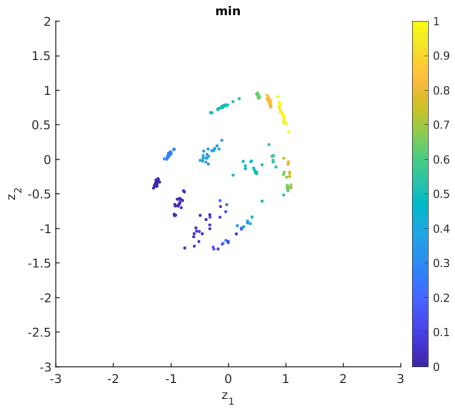
Table 7.2: Average performance metric with SVM accuracy and parameters for the GBPP.

7.1.5 Feature distribution

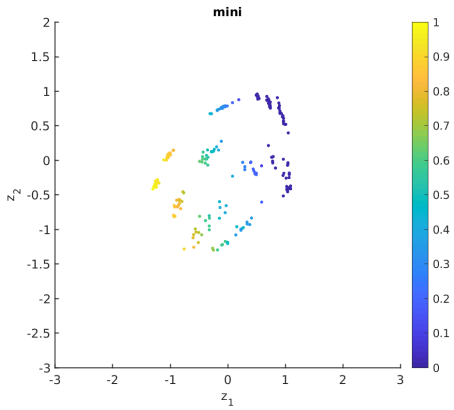
The features selected for explaining algorithm performance (variance, min, and mini) provide insight into why certain algorithms perform better in specific regions of the instance space. The top-left region of the instance space has low variance in the volume of items, meaning items in these instances have similar item sizes, making them easier for algorithms to fit them efficiently into bins. This explains why the algorithm like the nGreedy algorithm performs better in such region as it relies on such homogeneity. A low min values is found in the bottom-left region. When the smallest item has a low volume, it can influence how algorithms handle packing or binning tasks, as smaller items can fit into bins more easily and with less



(a) A 2D scatter plot of the instances based on the distribution of the var feature, which increases towards the bottom right.



(b) A 2D scatter plot of the instances based on the distribution of the min feature, which increases towards the top right.



(c) A 2D scatter plot of the instances based on the distribution of the mini feature, which increases towards the left.

Figure 7.5: A 2D scatter plot of the instances based on the distribution of important features, with purple indicating low values of the feature, and yellow indicating high values.

space left over. The GRASPm and LNS_Local are optimized to handle such cases effectively. Finally, a low mini value is found in the top-right region. This feature indicates the proportion of small items relative to the total number of items. A higher value means there is a higher proportion of small items, which can influence how well algorithms manage packing, especially those targeting specific types of items. The HS1m excels in such region, where there are many small items that can be easily squeezed into one bin, resulting in an efficient packing solution. The lack of instance diversity in the analysis, however, limits the ability to generalize the performance patterns of other algorithms across the entire instance space. To gain a better understanding, it would be beneficial to explore more diverse instances and their interactions with the algorithms.

7.2 Results from the updated instance space analysis

The iterative nature of ISA allows for adjustments to the meta-data and instance space in order to gain more insights and improve performance. The lack of diversity in the initial set of instances might have limited the ability to fully understand the algorithms' behavior. Therefore, the class gen presented in Section 5.1 were added into the instance space to increase

diversity and enable better performance evaluation.

7.2.1 Feature selection

The same 11 features were considered, and the optimal subset to predict good and bad performance was narrowed down to 5 features. The chosen features were: median, variance, max, min and mini. The projection into the 2D instance space is given below:

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} 0.6169 & 0.2898 \\ -0.0829 & 0.4929 \\ -0.1649 & 0.6187 \\ 0.0652 & -0.2031 \\ -0.4319 & -0.0117 \end{bmatrix}^T \begin{bmatrix} median \\ var \\ max \\ min \\ mini \end{bmatrix}$$

where Z_1 and Z_2 are the principal components.

7.2.2 Data visualization

The instances were scattered well in the instance space. This was to alienate the problem of different classes of instances re-using instances, causing a lack of diversity. Figure 7.6 shows how the instances are distributed around the instance space. The generated instances are densely scattered in the middle region of the instance space, overlapping a bit with some of the original instances. However, this was not a problem as the instances were generated using characteristics of the original instances. Thus, the insight they give should be useful for original instances.

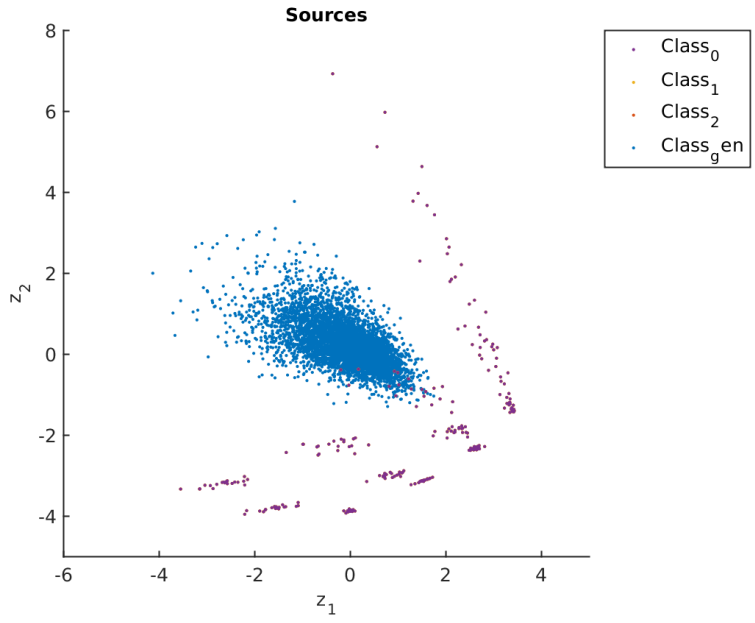
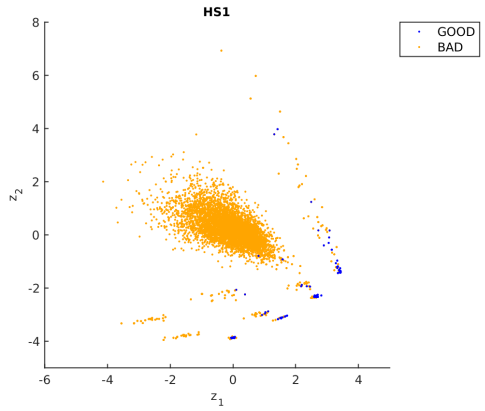
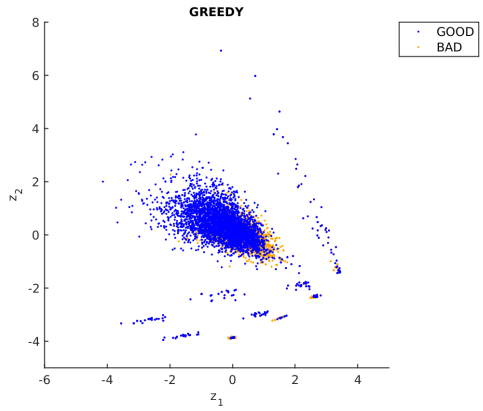


Figure 7.6: A 2D scatter plot of all data instances distributed by source based on the features selected in 7.2.1.

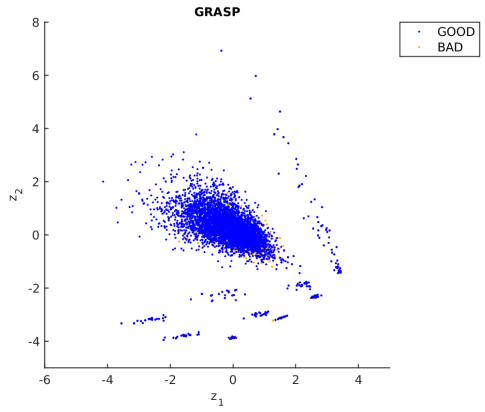
7.2.3 Performance distribution



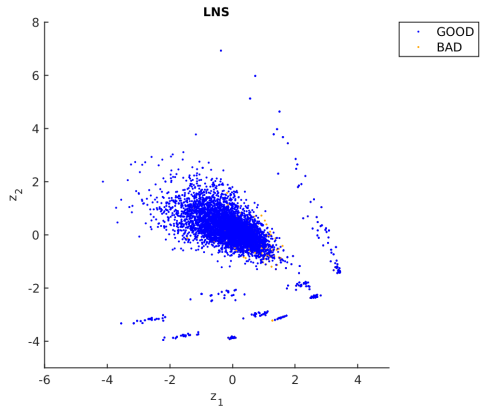
(a) Performance distribution of the HS1m algorithm over the instance space, with blue showing good performance and orange showing bad performance



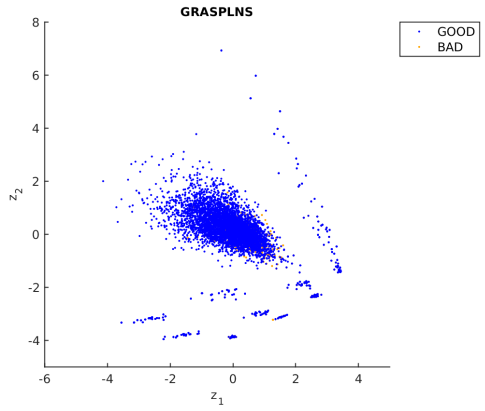
(b) Performance distribution of the nGreedy algorithm over the instance space, with blue showing good performance and orange showing bad performance.



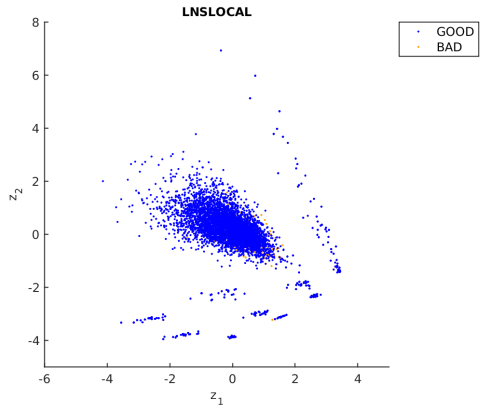
(c) Performance distribution of the GRASPm algorithm over the instance space, with blue showing good performance and orange showing bad performance.



(d) Performance distribution of the LNSm algorithm over the instance space, with blue showing good performance and orange showing bad performance.



(e) Performance distribution of the GRASP_LNS algorithm over the instance space, with blue showing good performance and orange showing bad performance.



(f) Performance distribution of the LNS_Local algorithm over the instance space, with blue showing good performance and orange showing bad performance.

Figure 7.7: Performance distribution of the six proposed algorithms over the instance space, with blue showing good performance and orange showing bad performance.

The analysis of the performance distribution in Figure 7.7 and Table 7.3 provide valuable insight into the effectiveness of the different algorithms across the instances. As shown in Figure 7.7, instances where the algorithms perform well are highlighted in blue, while orange indicates poor performance. The algorithms tend to perform well in most instances, with the occasional exception where their performance degrades. However, the poor performance instances are scattered across the instance space and not clustered in any one area, suggesting that there isn't a specific instance type or class where an algorithm consistently fails. In particular, the HS1m algorithm has the lowest percentage of good performance, with only 2.8% of instances where it performs well. The U-FFD and U-BFD algorithms perform relatively well, achieving 76.7% and 81.9% good performance, respectively. The nGreedy algorithm follows closely with 68.9% of instances where it performs well, which indicates it is fairly competitive with the benchmarks. The GRASPm, LNSm, GRASP_LNS, and LNS_local show excellent performance, with the percentage of good performance ranging from 85.2% to 89.5%. Among these, the LNSm and LNS_Local algorithms perform the best.

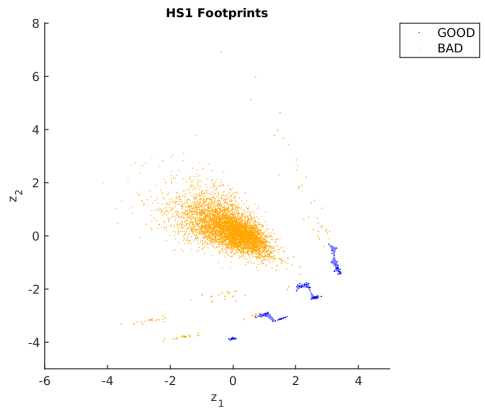
Algorithm	Actual % of good performance
U-FFD	76.7%
U-BFD	81.9%
HS1m	2.8%
nGreedy	68.9%
GRASPm	85.2%
LNSm	89.5%
GRASP_LNS	88.9%
LNS_Local	89.5%

Table 7.3: Performance distribution table indicating the percentage of good and bad performance for each algorithm with respect to original and generated data instances.

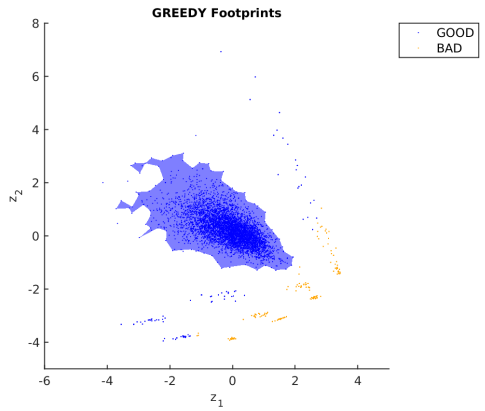
7.2.4 Algorithm footprints

In this section, the performance of the different algorithms is visually explored within the instance space, and the SVM selector embedded in MATILDA is used to recommend the most suitable algorithm for each region. As shown in Figure 7.8, the HS1m is expected to perform well in the bottom-right region of the instance space. This suggests that HS1m might be optimized for specific types of instances, particularly those with certain characteristics. The nGreedy, GRASPm, LNSm, GRASP_LNS and LNS_Local algorithms are expected to perform well everywhere else except the bottom-right region, which might correspond to instances that are easier to solve. This suggests that they are more versatile across various instance types. Most of the algorithms showing almost similar regions of predicted good performance. However, the HS1m was predicted to perform good in regions where the others were predicted to perform bad. Thus, as bad as its performance was, it still had regions were it was predominantly better than the rest.

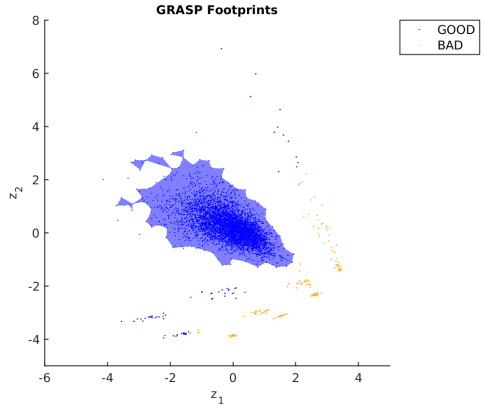
The results suggest that there are five algorithms best equiped to handle instances all across the instance space. The LNSm was the algorithm recommended in the majority of the instance space, with the HS1m, nGreedy, GRASPm and GRASP_LNS also having regions of recommendation. This is supported by Table 7.4 which shows the performance of the SVM



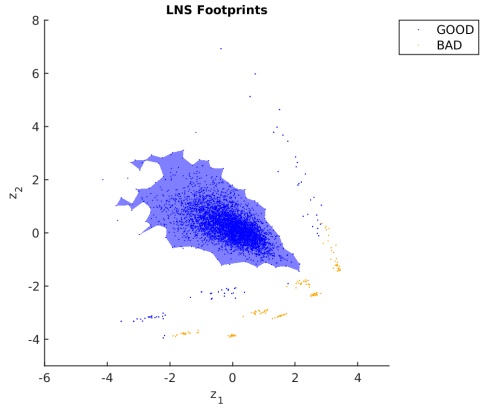
(a) Performance prediction of the HS1m algorithm over the updated instance space, with the region of expected strong performance highlighted in blue.



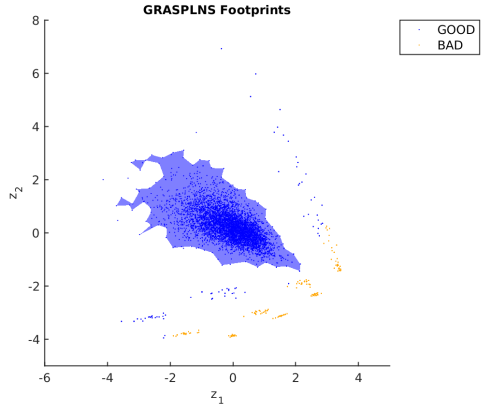
(b) Performance prediction of the nGreedy algorithm over the updated instance space, with the region of expected strong performance highlighted in blue.



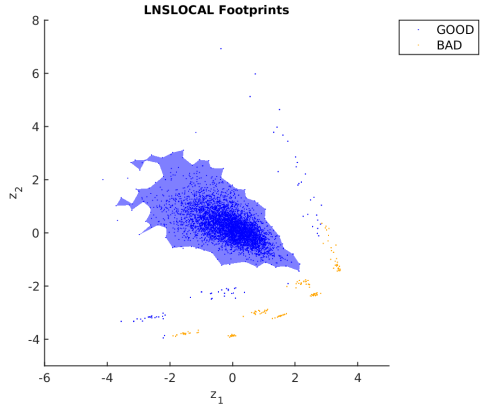
(c) Performance prediction of the GRASPm algorithm over the updated instance space, with the region of expected strong performance highlighted in blue.



(d) Performance prediction of the LNSm algorithm over the updated instance space, with the region of expected strong performance highlighted in blue.



(e) Performance prediction of the GRASP_LNS algorithm over the updated instance space, with the region of expected strong performance highlighted in blue.



(f) Performance prediction of the LNS_Local algorithm over the updated instance space, with the region of expected strong performance highlighted in blue.

Figure 7.8: Performance predictions of the six proposed algorithms over the updated instance space, with the region of expected strong performance highlighted in blue.

selector, together with SVM accuracy, precision and recall. Thus, It shows how often the

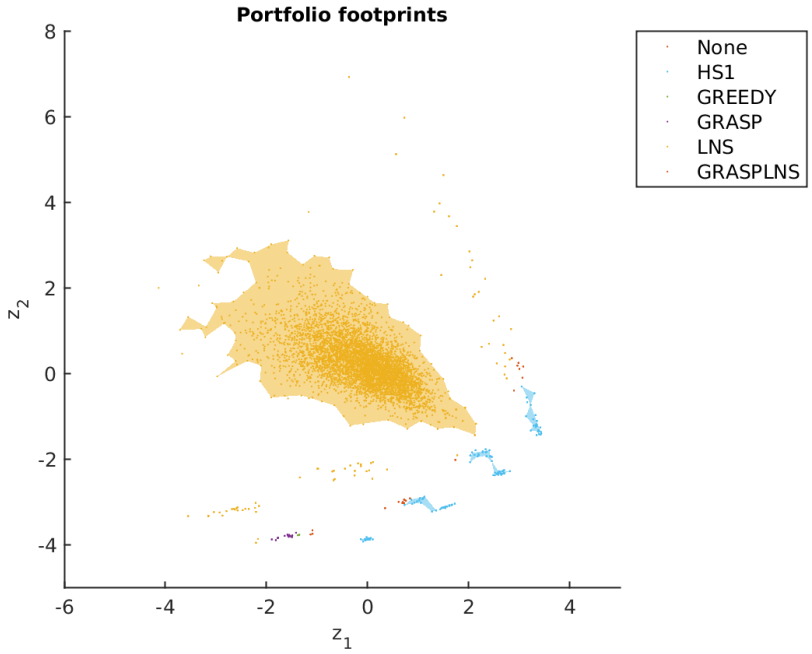


Figure 7.9: The SVM recommendation of algorithms to use for each region of the instance space.

SVM chooses an algorithm with good performance.

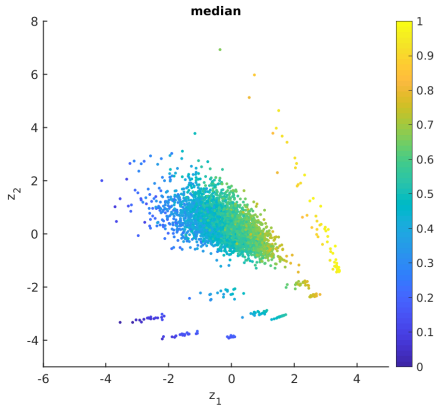
Algorithm	Accuracy	precision	recall
U-FFD	76.7%	79.1%	94.7%
U-BFD	81.8%	85.1%	94.4%
HS1m	95.9%	38.3%	91.2%
nGreedy	69.6%	70.9%	94.6%
GRASPm	81.3%	86.3%	92.9%
LNSm	85.4%	90.7%	93.3%
GRASP_LNS	84.6%	90%	93%
LNS_Local	85.4%	90.7%	93.3%
Selector	N/A	87.5%	N/A

Table 7.4: Average performance metric with SVM accuracy and parameters for the GBPP.

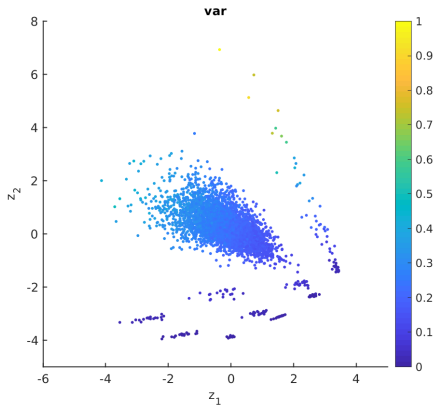
7.2.5 Feature distribution

The features selected for explaining algorithm performance (median, variance, max, min, and mini) provide insight into why certain algorithms perform better in specific regions of the instance space. The bottom-right region of the instance space has a low mini and low variance in the volume of items. The low mini indicates the proportion of small items relative to the total number of items while a low variance means that items in these instances have similar item sizes, making them easier for algorithms to fit them efficiently into bins. The HS1m excels in such regions, where there are many small items that can be easily squeezed into one bin. A low min values is found in the top-left region. When the smallest item has a low volume, it can influence how algorithms handle packing or binning tasks, as smaller items can fit into bins more easily and with less space left over. The nGreedy, GRASPm, LNSm,

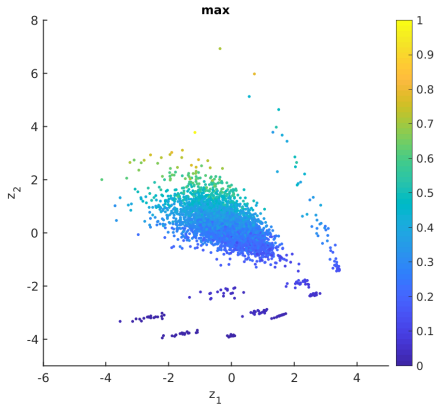
GRASP_LNS and LNS_Local proved to handle such cases effectively.



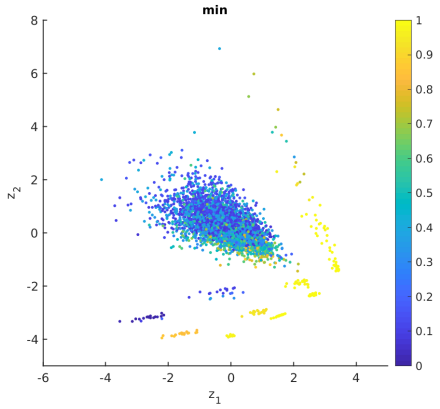
(a) A 2D scatter plot of the instances based on the distribution of the median feature, which increases towards the right.



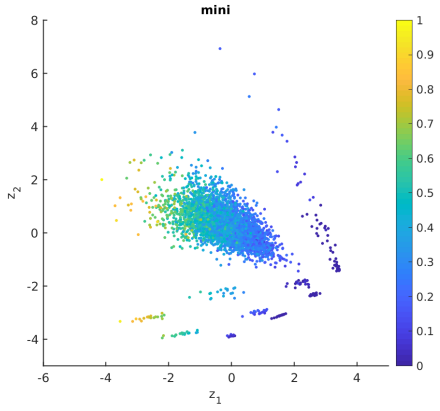
(b) A 2D scatter plot of the instances based on the distribution of the var feature, which increases towards the top-left.



(c) A 2D scatter plot of the instances based on the distribution of the max feature, which increases towards the top.



(d) A 2D scatter plot of the instances based on the distribution of the min feature, which increases towards the bottom.



(e) A 2D scatter plot of the instances based on the distribution of the mini feature, which increases towards the left.

Figure 7.10: A 2D scatter plot of the instances based on the distribution of important features, with purple indicating low values of a feature and yellow indicating high values.

CHAPTER 8

DISCUSSION

In this chapter, we will revisit the results presented in Chapters 6 and 7, and draw conclusions based on the findings. The primary focus will be on discussing the average performance of the algorithms, followed by a deeper exploration of the insights gained beyond just average performance. This will provide a more holistic understanding of the algorithms' effectiveness, how they can be interpreted, and what factors contribute to their performance.

8.1 Algorithm performance evaluation

The results presented in Chapter 6 demonstrate a marked improvement in the percentage gap between the upper and lower bounds across the proposed algorithms. Four out of the six proposed algorithms—GRASPm, LNSm, GRASP LNS, and LNS Local—were able to reduce the percentage gap relative to the U-FFD and U-BFD benchmarks, outperforming them in this regard. On average, the algorithms performed with a gap of 12.384%, with the GRASP_LNS showing a notably lower percentage gap (7.976%) compared to others. The GRASP_LNS algorithm consistently performed as one of the best algorithms, closely followed by the GRASPm, LNSm, and LNS_local. The nGreedy algorithm performed similarly to the benchmark heuristics, indicating it did not have a strong advantage over the U-FFD and U-BFD, but also did not perform badly. The HS1m algorithm was the worst-performing algorithm, as it not only failed to reduce the percentage gap, but also increased it significantly in comparison with the other algorithms where the percentage gaps are not too far apart.

The most notable reduction in the percentage gap occurred when all items were considered non-compulsory. This observation suggests that the algorithms are particularly effective at minimizing the percentage gap when they have the flexibility to decide which items to pack, as opposed to when all items must be packed. In cases where items are compulsory, the performance of the algorithms was similar, with the exception of the HS1m, which consistently underperformed. Overall, four algorithms managed to approach the lower bound more closely than the benchmark, with GRASP_LNS emerging as the top performer in terms of reducing the gap to the lower bound. Additionally, the number of items in an instance did not significantly affect the performance gap between the algorithms, indicating that the heuristics were robust and consistent across different sizes of problems. This was particularly important because

it meant that the algorithms could perform well regardless of the complexity added by the number of items.

The Nemenyi test performed on the results showed that statistical significance was minimal across most of the algorithms when compared to the benchmark algorithms. The HS1m was the only algorithm to show a statistically significant difference, and this was primarily due to its poor performance on several instances, rather than any clear improvement over the others. The GRASP_LNS came close to showing a statistically significant improvement, but this was not enough to decisively reject the null hypothesis. However, the GRASP_LNS still performed better overall, with the lowest gap to the lower bound and high consistency across different instances.

Further analysis was conducted across different classes of instances. The Friedman test confirmed that the algorithms did not yield equivalent results, even in cases where boxplots suggested minimal variation. A p-value of less than 2.2^{-16} indicated a highly significant difference, leading to the rejection of the null hypothesis that the heuristics produced equivalent results in terms of net cost. This analysis demonstrated that the heuristics do not return equivalent results, thus further emphasizing their distinct performance profiles. Class 1, which contains non-compulsory items, exhibited greater statistical significance in the performance differences between algorithms when compared to Class 0, which consists only of compulsory items. In Class 0, the algorithms' performances were relatively similar, as the best achievable outcome was marginally reducing the number of bins used, which was not substantial enough to be statistically significant. Conversely, in Class 1, where items are non-compulsory, the algorithms showed significantly different performance levels. Additionally, the Nemenyi test indicated no improvement from performing a local search following a large neighbourhood search, as seen in the results for the LNSm and LNS_local. This suggests that the large neighbourhood search in LNSm was already highly efficient and further attempts to substitute packed items with unpacked ones did not yield additional improvements in net cost.

Given the improvement observed in the algorithms' performance compared to the U-FFD and U-BFD benchmarks, the key question became: "What further insights can be derived beyond average performance?" To address this, the ISA was employed, initially using instances from the literature and later incorporating generated instances to gain deeper insights into the problem.

8.2 Instance space analysis evaluation

The ISA framework highlighted the significance of selected features in explaining the difficulty of instances for various algorithms, as outlined in Chapter 7. The chosen features—var, min, and mini—worked synergistically to predict algorithmic performance. As the value of each feature increased, it became progressively less effective at predicting algorithm performance. The selected features, however, spanned different regions of the instance space, enabling them to collectively explain algorithmic performance across diverse conditions. In regions where certain features struggled to account for algorithmic performance, at least one feature was consistently able to provide clarity, thus justifying the rationale behind feature selection.

In the initial instance space, variance was particularly effective at explaining algorithm

performance in the upper-right region, while minimum was more effective in the lower-left region, and mini excelled along the right side. Together, these features complemented each other in offering a comprehensive explanation of algorithm performance. The GRASP_LNS algorithm demonstrated the best performance, achieving a high success rate of 93.5% across instances, positioning it as the algorithm that could be expected to perform well throughout the instance space. In contrast, the HS1m algorithm displayed the poorest performance, with a success rate of just 35.9%. The remaining algorithms exhibited varying degrees of performance, with their success rates ranging from 69.5% to 91.8%. This variation in performance across algorithms offers a valuable comparative insight, suggesting that each algorithm is suited to different regions within the instance space.

The SVM selector identified the GRASP_LNS as the algorithm that performed well 93.5% of the time, aligning precisely with its actual success rate. This observation underscores why GRASP_LNS was predicted to excel across the entire instance space. With sufficient diversity in the instances, the SVM should, in theory, predict different algorithms to perform optimally in distinct regions of the instance space, resulting in a portfolio footprint that incorporates multiple algorithms rather than relying solely on one. The SVM selector also exhibited the highest precision for the GRASP_LNS, highlighting its ability to consistently recommend this algorithm, even in regions where other algorithms also performed well. This suggests that while GRASP_LNS algorithm may dominate the recommendations in the portfolio footprint, it does not imply that other algorithms are incapable of achieving good results in specific regions.

To enhance the diversity of the instance space, additional instances were generated and projected into the instance space, leading to the identification of five key features sufficient to explain algorithmic performance. Notably, the 'median', 'var', 'max', 'min', and 'mini' all had specific regions in the instance space in which they struggled to explain performance. Nevertheless, even in regions where some features faltered, there was always at least one feature capable of explaining the algorithm's performance

The LNSm algorithm achieved the best performance overall, performing well on 89.5% of instances, whereas the HS1m algorithm exhibited the poorest performance, with only 2.8% of successful instances. Despite its overall subpar performance, the HS1m algorithm was still recommended in certain regions where it outperformed all other algorithms. This demonstrates that an algorithm's poor average performance does not preclude it from excelling in specific areas of the instance space. Similarly, while the LNSm algorithm achieved the highest overall performance, it still displayed some regions of suboptimal performance.

The SVM selector aggregated the predictions from all algorithms to generate an overall portfolio footprint, which highlighted the regions within the instance space where each algorithm was recommended. The selector predicted that five algorithms would perform well in distinct regions, leaving certain areas where no algorithm was expected to perform optimally. The LNSm algorithm, with the highest precision of 90.7%, was forecasted to perform well in the largest area of the instance space, making it the preferred choice in regions where multiple algorithms excelled. Interestingly, despite its overall poor performance, the HS1m algorithm was recommended in regions where it outperformed all other algorithms. This finding illustrates the potential risk of prematurely dismissing an algorithm based solely on

its average performance, as doing so may result in missed opportunities where that algorithm excels in certain areas. Therefore, even algorithms with lower average performance can still be valuable under specific conditions. Finally, the SVM selector demonstrated an actual success rate of 87.5%, indicating that it accurately selected the best-performing algorithm 87.5% of the time. This level of precision implies that the selector's predictions are incorrect only 12.5% of the time, which is considered sufficient for the scope of this study.

In summary, the results from Chapter 6 and Chapter 7 show that while average performance gives a useful overview, a deeper analysis based on instance-specific behaviour and feature distributions can provide richer insights into why and where algorithms perform well. This understanding can guide the selection of the best algorithm for specific problems, ensuring that the right tool is chosen for the task at hand. The code used for this project is available on the following GitHub repository [11].

CHAPTER 9

CONCLUSION

This chapter contains a summary of the research documented in this thesis, highlighting in Section 9.1 how the objectives in Section 1.3 are accomplished, and culminating in an appraisal in Section 9.2 of the novel contributions made in this thesis. Finally, suggestions for possible future work as follow-up work on the research carried out in this thesis will be described briefly in Section 9.3.

9.1 Summary of the thesis

The introductory chapter provided a concise overview of the problem at hand and underscored its real-world significance. This was followed by a clear articulation of the research aim, along with the specific research objectives to be pursued. The chapter concluded with an outline of the thesis structure, providing the reader with a roadmap for the subsequent sections.

Part I of the thesis was dedicated to reviewing the literature surrounding the GBPP and its variants, as well as the literature ISA. This part was essential for achieving objectives *i* and *ii* of the thesis. Chapter 2, the first chapter of Part I, offered a comprehensive overview of the GBPP. The chapter began by discussing prominent typologies of packing problems, which provided important context for understanding the characteristics of the GBPP. It then outlined various variants of the GBPP that have emerged over time, highlighting their differences from the original problem setting. Additionally, the chapter discussed the different solution approaches employed to tackle the GBPP and its variants, setting the stage for addressing objective *iii*. Finally, the characteristics of the GBPP were defined with respect to the relevant typology.

In Chapter 3, the second chapter of Part I, the focus shifted to ISA, providing an overview of its general framework. This chapter aimed to demonstrate how ISA compares algorithms based on relevant meta-data. A detailed background of ISA and its development was presented, along with an examination of other packing problems that have utilized ISA, accompanied by a brief discussion of the results obtained in those studies. The chapter concluded with an outline of the specific focus of this thesis in relation to ISA.

Part II commenced with a detailed description of the heuristics proposed in the thesis. Each

heuristic was described thoroughly, with the use of pseudo-codes and illustrative examples to demonstrate how the algorithms tackle the packing of items into bins. Benchmark instances were used to evaluate the performance of these algorithms. The heuristics proposed in this chapter included the HS1m, nGreedy, GRASPm, LNSm, GRASP_LNS, and LNS_Local, all of which were designed to challenge the U-FFD and U-BFD algorithms from the original GBPP literature.

Chapter 5 provided a detailed explanation of the ISA framework and how it was applied to the GBPP in this research. The meta-data used for the ISA analysis were meticulously detailed, including the data instances, the features calculated for the algorithms (which were described in the previous chapter), and the performance metrics used for comparing the algorithms.

In Chapter 6, which marked the beginning of Part III, the computational results of the proposed heuristics, as well as those from the original literature, were presented. These results were displayed in the form of boxplots, with supplementary tables containing statistical test results to determine if there were significant differences between the proposed algorithms and the benchmark algorithms. The tables also included results from statistical tests that ensured the algorithms adhered to the constraints, particularly with regard to not violating the lower bound.

Chapter 7 presented the results from the initial ISA, showing the distribution of various performance measures within the instance space. Figures were used to visualize the actual performance of the algorithms in comparison to their predicted performance, illustrating the regions where each algorithm performed well or poorly. This visualization aimed to predict the optimal algorithm to use for a given random instance. A table was also included, showing the percentage performance for each algorithm, as well as the performance of the selector, which identified the best-performing algorithm across the instance space. To address the lack of diversity in the initial instance space, additional instances were generated and projected into the instance space. The same result presentation format was repeated to gain further insights from these additional instances, which revealed new patterns not apparent in the original instance space.

The results presented in the earlier chapters of Part III were thoroughly discussed in Chapter 8, the final chapter of Part III. This chapter provided an in-depth analysis of the implications of the results and summarized the key conclusions drawn from the study:

- The U-FFD and U-BFD algorithms produced excellent results when applied to the data instances from the literature.
- Most of the proposed algorithms outperformed the U-FFD and U-BFD algorithms, with the HS1m algorithm showing the worst performance, while the GRASP_LNS algorithm achieved the best average performance.
- Four of the proposed algorithms reduced the gap to the lower bound achieved by U-FFD and U-BFD.
- All algorithms adhered to the packing constraints, ensuring that they did not violate the lower bound in their solutions.

- Even the best-performing algorithms exhibited regions of weak performance, demonstrating that an algorithm's average performance does not guarantee optimal results across the entire instance space.
- Conversely, the worst-performing algorithm on average, HS1m, still showed regions of good performance and was the top performer in those regions.
- The LNSm algorithm was identified as having the largest region where it was predicted to perform best, although there were small regions in the instance space where no algorithm was predicted to perform well.
- Overall, five of the proposed algorithms complemented each other in the ISA selection process, each being the preferred algorithm in specific regions of the instance space

The chapter concluded by emphasizing the critical role of the ISA in identifying the optimal algorithm to use in various regions of the instance space. By selecting the most appropriate algorithm for each region, one can achieve better results than relying on the best algorithm on average. This approach has the potential to enhance efficiency and improve outcomes for packing solutions.

9.2 Appraisal of contributions

Contribution 1: *A review of existing heuristics for the GBPP and its variants.*

A primary objective of this thesis was to propose and evaluate heuristics aimed at improving the performance of algorithms dealing with the GBPP. To set the stage for the proposed solutions, a thorough review of the existing literature on the GBPP and its variants was conducted. In the review, the problem was contextualized within the broader framework of bin packing problems. The various heuristic and exact methods that have been employed to solve the GBPP over the years served as an essential foundation for the development of new heuristics to challenge the existing solutions, especially the basic U-FFD and U-BFD algorithms, which have become standard approaches in the field.

Contribution 2: *A review of instance space analysis with respect to packing problems.*

Another key contribution of this thesis was the application of ISA to the GBPP and the algorithms designed to solve it. To facilitate this, an extensive review was conducted on the use of ISA in packing problems. ISA, as a method for understanding algorithm performance across diverse instance spaces, has been widely used in the context of other optimization problems. The review highlighted the relevance of ISA in comparing different algorithms based on their performance across various regions of the instance space. By understanding how ISA had been applied in related fields, this thesis was able to position the GBPP within this context and leverage ISA to offer deeper insights into the performance of the proposed heuristics.

Contribution 3: *Proposal of new heuristics for the GBPP.*

Building on the previous contributions, the thesis proposed new heuristics for solving the GBPP, drawing inspiration from existing methods used for similar packing problems. This contribution arose from a gap identified in the literature, which lacked heuristics designed

to effectively challenge the basic U-FFD and U-BFD algorithms. The proposed algorithms aimed to enhance the solution quality by incorporating different strategies and improving the basic heuristics. These six new heuristics were designed with the goal of offering more robust solutions to the GBPP.

Contribution 4: *A statistical comparison of the relative performances of eight heuristics in terms of solution quality.*

To assess the efficacy of the proposed heuristics, a comprehensive computational study was conducted. Six newly proposed algorithms were compared with the two basic algorithms (U-FFD and U-BFD) from the literature. Statistical tests were performed using a 95% confidence interval to evaluate the differences in solution quality across the algorithms. The results indicated that, on average, the proposed algorithms did not show drastic differences from the benchmark algorithms in overall performance. However, when the performance was broken down into distinct classes, some algorithms demonstrated significant improvements. Notably, all algorithms showed a significant difference when compared to the lower bound, confirming that none of the algorithms violated the packing constraints, thus maintaining the integrity of the problem's solution space.

Contribution 5: *Application of ISA to the GBPP heuristics for the first time.*

The final major contribution of this thesis was the application of ISA to compare the performance of the heuristics in a way that went beyond traditional average performance metrics. In this analysis, the proposed algorithms were compared in terms of their strengths and weaknesses across different regions of the instance space. The ISA framework was applied using the data instances from the literature, and the results were initially revealing. The performance of the algorithms was not only examined in terms of their average outcomes but also in terms of their predicted performance across various regions of the instance space. This allowed for a more nuanced understanding of where each algorithm excels or falters. Furthermore, additional instances were generated to enrich the instance space, offering further insights that the initial dataset could not provide. The results highlighted that, while the GRASP_LNS algorithm performed best on average, the ISA framework predicted that the LNSm would perform the best across a larger portion of the instance space, offering a deeper insight into algorithm selection.

9.3 Suggestions for future work

Suggestion 1: *Testing performance of more algorithms in ISA.*

As packing problems continue to be a vibrant area of research, many new algorithms have been proposed across various problem domains. Although the current focus of this thesis has been on the GBPP, many of these algorithms—while not specifically designed for GBPP—could potentially be adapted and tested within this context. By incorporating more algorithms into the ISA, it would be possible to compare their performance against the benchmark algorithms used in this study. Testing additional algorithms could uncover more efficient approaches or reveal new strategies that perform better under certain conditions, thereby further advancing the state of the art in solving the GBPP. This would enrich the ISA framework and allow for

a more comprehensive evaluation of algorithmic performance across diverse solution spaces.

Suggestion 2: *Generate more instances that are better suited for the ISA.*

The diversity of instances used in this study has been somewhat limited, particularly in the context of the GBPP. While generating random instances that better reflect real-world scenarios is challenging, doing so would significantly enhance the value of the instance space analysis. More diverse instances, representing a broader range of practical packing scenarios, would allow for a more accurate understanding of how the proposed heuristics perform in varied conditions. By expanding the instance space, the analysis could also reveal previously unnoticed patterns in algorithm performance, providing more detailed insights into which algorithms are most effective for specific problem instances. A broader problem space would not only improve the predictive power of the ISA but would also make the research more applicable to real-world problems.

Suggestion 3: *Use of real-world data.*

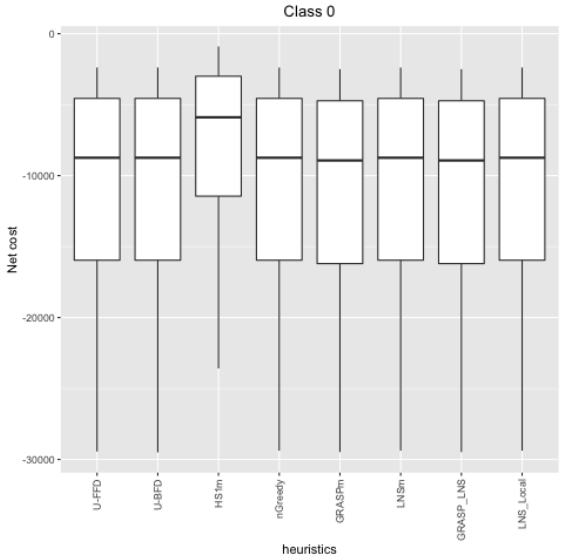
In the context of this thesis, the data instances from Baldi *et al.*'s repository [16] were employed to test the proposed algorithms. While this repository provided a useful starting point, it is important to note that the problem instances used were hypothetical. To increase the relevance of the results to practical applications, real-world data should be used for testing the proposed heuristics. Logistics companies, for example, may encounter additional constraints or unique conditions not reflected in the simulated instances used in this study. Incorporating real-world data would help to refine the algorithms, ensuring they are better suited to the complexities and operational needs of businesses in the logistics and packing industries. Such data could also offer insights into how the algorithms could be adapted to handle specific operational constraints, making the solutions more practically adoptable by industry stakeholders.

Suggestion 4: *Solving higher dimensions of the problem.*

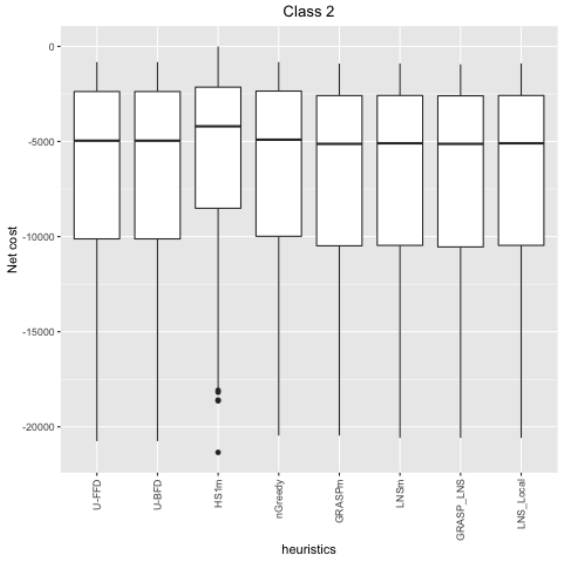
This thesis focused on a one-dimensional version of the GBPP, where the primary characteristic used to calculate features was the volume of items being packed. However, in real-world applications, the problem might involve additional dimensions. For instance, packing problems could require consideration of not only the volume of items but also their complex geometrical properties such as length, breadth, and height. The introduction of multi-dimensionality into the problem would allow for the exploration of more complex features and the development of more sophisticated algorithms. It would be particularly interesting to investigate how many features are required to explain algorithm performance effectively without redundancy. The inclusion of more dimensions could lead to new insights about the trade-offs involved in packing items with multiple attributes, potentially improving algorithmic performance in more complex, real-world settings.

APPENDIX A

ADDITIONAL BOXPLOTS AND TESTS



(a) Boxplot showcasing the net cost achieved by each of the eight heuristics presented in Chapter 4 with respect to class 0 data instances explained in section 5.1.



(b) Boxplot showcasing the net cost achieved by each of the eight heuristics presented in Chapter 4 with respect to class 2 data instances explained in section 5.1.

Figure A.1: Boxplot showcasing the net cost achieved by each of the eight heuristics presented in Chapter 4 with respect to class 2 data instances explained in section 5.1.

Figure A.2 shows the percentage gap from the upper bound to the lower bound achieved by all the algorithms considered in this thesis, with respect to class 0 and class 2 instances explained in Section 5.1.

Figure A.3 shows the percentage gap from the upper bound to the lower bound achieved by all the algorithms considered in this thesis, with respect to all instances containing 25, 50, 100 and 200 items explained in Section 5.1.

Figure A.1 shows the upper bound achieved by all the algorithms considered in this thesis, with respect to class 0 and class 2 instances explained in Section 5.1.

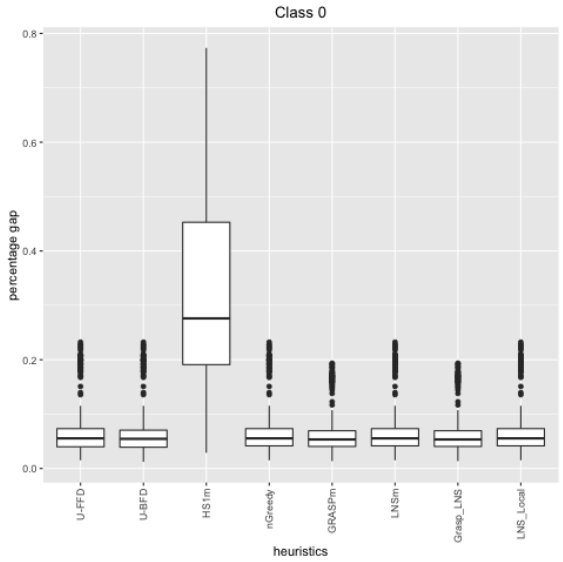
Algorithms	p-value (class 0)	p-value (class 2)
U-FFD and U-BFD	0.9420	0.9427
U-FFD and HS1m	$1.6e^{-06}***$	0.1189
U-FFD and nGreedy	0.9769	0.7830
U-FFD and GRASPm	0.8471	0.3513
U-FFD and LNSm	0.9769	0.3103
U-FFD and GRASP_LNS	0.8471	0.2887
U-FFD and LNS_Local	0.9769	0.3103

Table A.1: A comparison of results from the eight algorithms, with the benchmark for comparison being the U-FFD, and all the other algorithms being compared to it, for class 0 and class 2 data instances in 5.1.

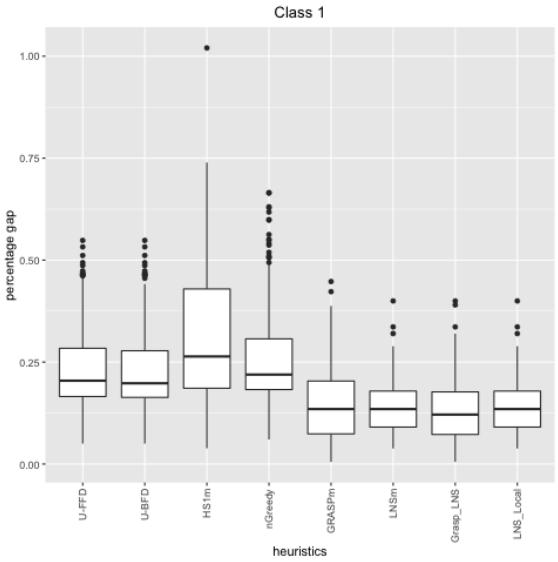
Table A.1 shows the results from an ad hoc test on the difference between algorithms in terms of net cost achieved.

Figure A.4 shows instances in the instance space where each algorithm performs bad on the original instances, to support the percentages of good performance given.

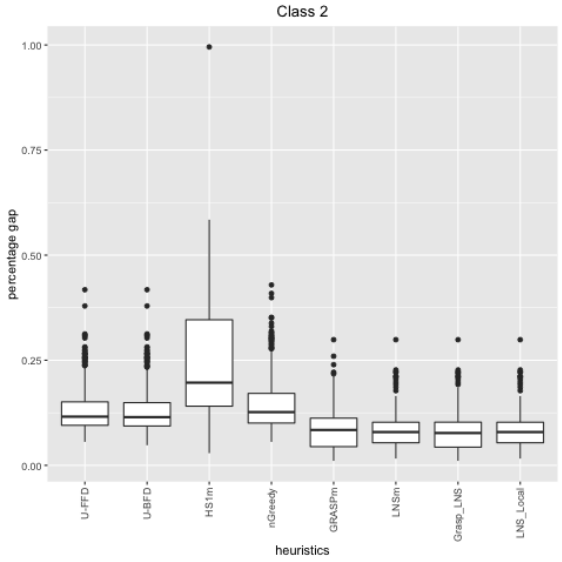
Figure A.5 shows instances in the instance space where each algorithm performs bad on the updated instance space, to support the percentages of good performance given.



(a) Boxplot showcasing the percentage gap achieved by each of the eight heuristics presented in Chapter 4 with respect to class 0 data instances explained in section 5.1.

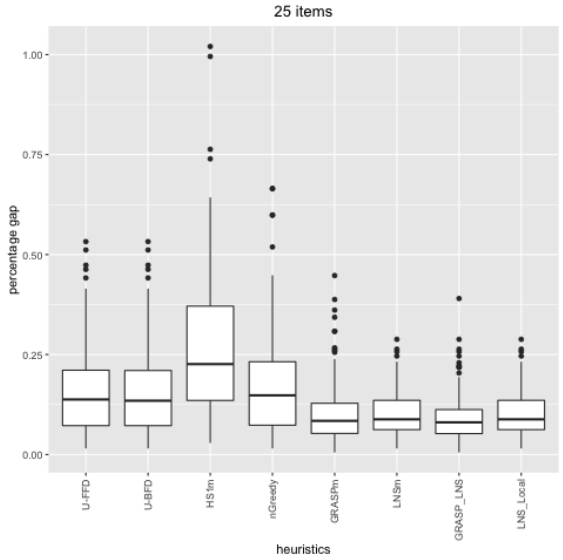


(b) Boxplot showcasing the percentage gap achieved by each of the eight heuristics presented in Chapter 4 with respect to class 1 data instances explained in section 5.1.

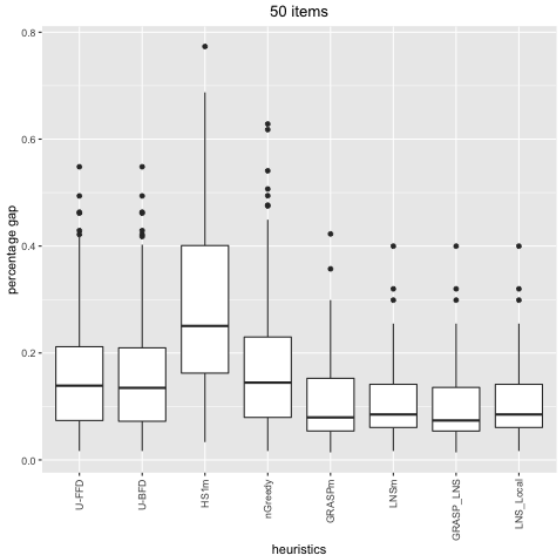


(c) Boxplot showcasing the percentage gap achieved by each of the eight heuristics presented in Chapter 4 with respect to class 2 data instances explained in section 5.1.

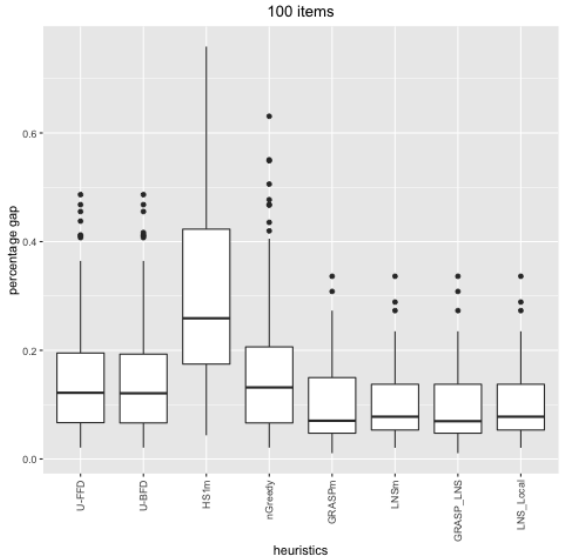
Figure A.2: Boxplot showcasing the percentage gap achieved by each of the eight heuristics presented in Chapter 4 with respect to class 0, class 1 and class 2 data instances explained in section 5.1.



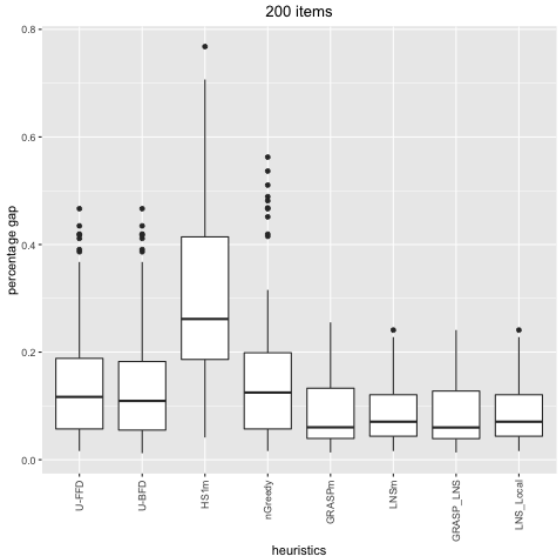
(a) Boxplot showcasing the percentage gap achieved by each of the eight heuristics presented in Chapter 4 with respect to instances with 25 items explained in section 5.1.



(b) Boxplot showcasing the percentage gap achieved by each of the eight heuristics presented in Chapter 4 with respect to instances with 50 items explained in section 5.1.

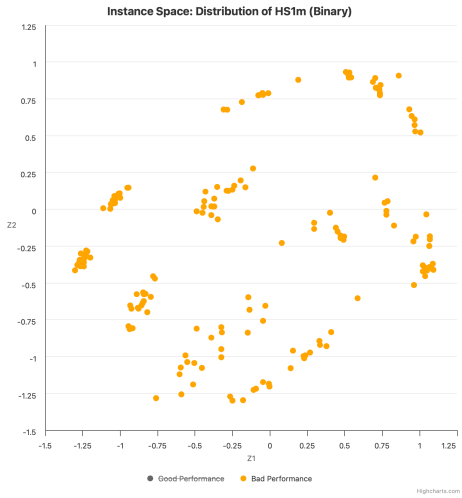


(c) Boxplot showcasing the percentage gap achieved by each of the eight heuristics presented in Chapter 4 with respect to instances with 100 items explained in section 5.1.

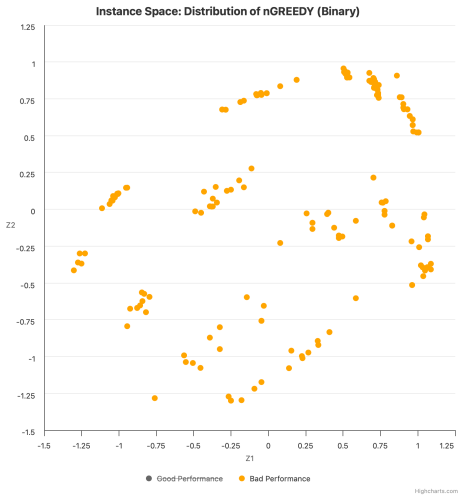


(d) Boxplot showcasing the percentage gap achieved by each of the eight heuristics presented in Chapter 4 with respect to instances with 200 items explained in section 5.1.

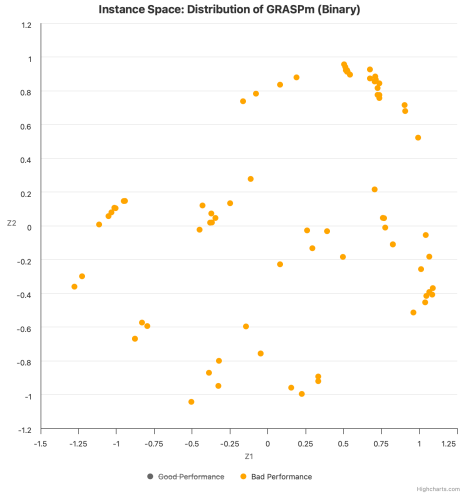
Figure A.3: Boxplot showcasing the percentage gap achieved by each of the eight heuristics presented in Chapter 4 with respect to all data instances containing 25, 50, 100, or 200 items.



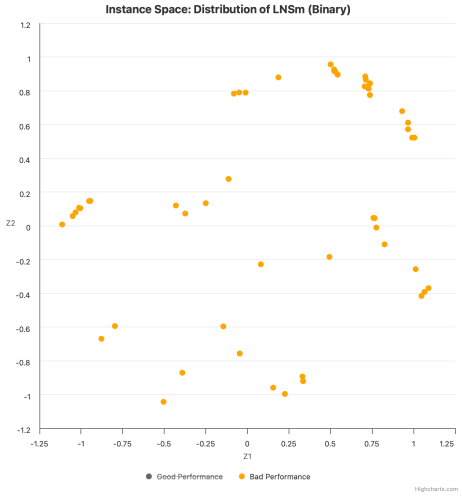
(a) Performance distribution of HS1m over the instance space, only showing bad performance.



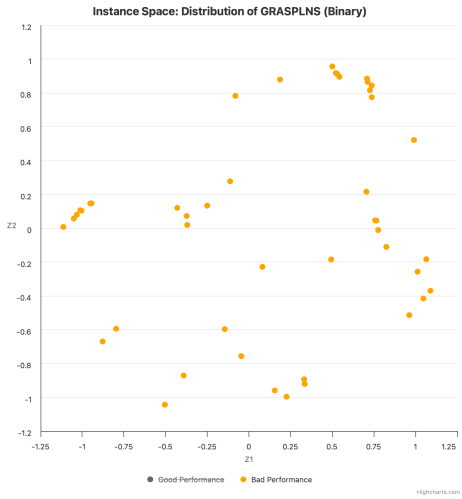
(b) Performance distribution of nGreedy over the instance space, only showing bad performance.



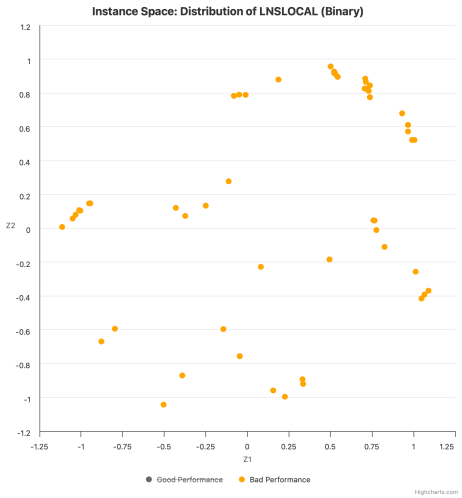
(c) Performance distribution of GRASPm over the instance space, only showing bad performance.



(d) Performance distribution of LNSm over the instance space, only showing bad performance.

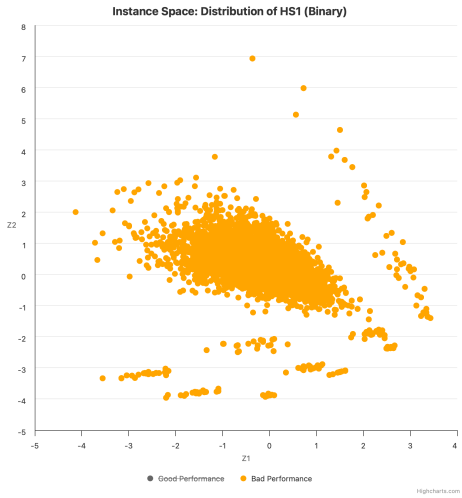


(e) Performance distribution of GRASP_LNS over the instance space, only showing bad performance.

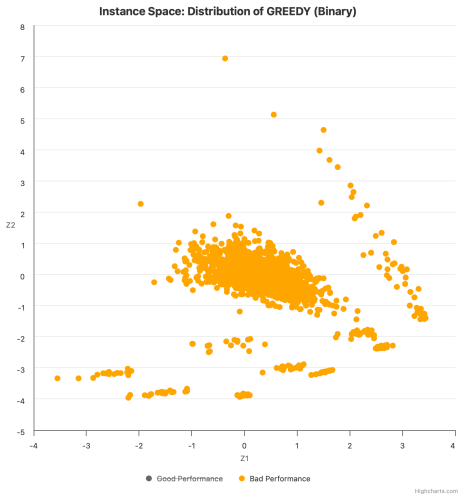


(f) Performance distribution of LNS_Local over the instance space, only showing bad performance.

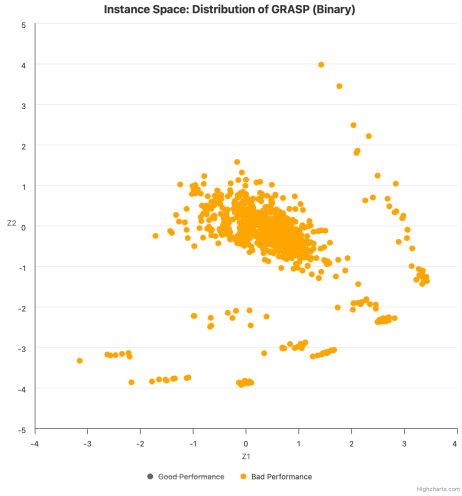
Figure A.4: Performance distribution of the six proposed algorithms over the instance space, only showing bad performance.



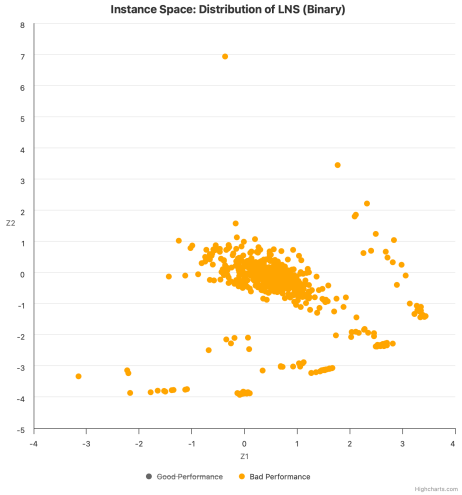
(a) Performance distribution of HS1m over the updated instance space, only showing bad performance.



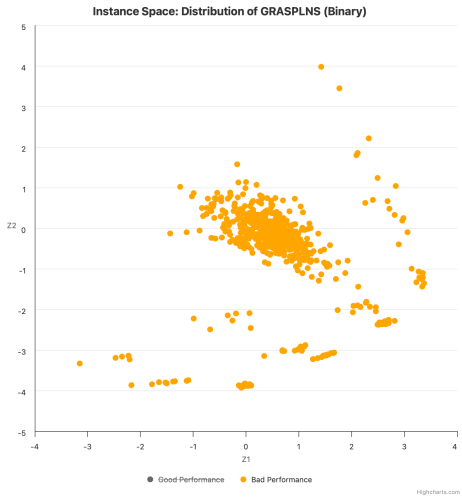
(b) Performance distribution of nGreedy over the updated instance space, only showing bad performance.



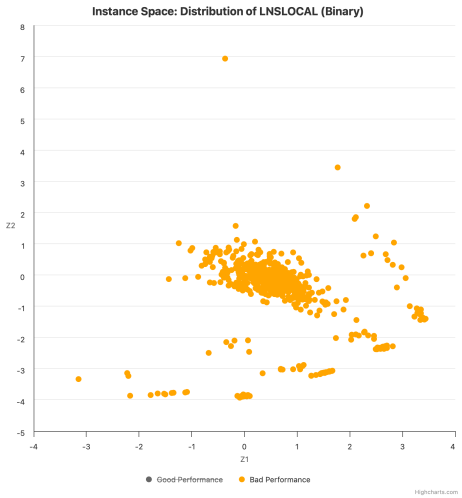
(c) Performance distribution of GRASPm over the updated instance space, only showing bad performance.



(d) Performance distribution of LNSm over the updated instance space, only showing bad performance.



(e) Performance distribution of GRASP_LNS over the updated instance space, only showing bad performance.



(f) Performance distribution of LNS_Local over the updated instance space, only showing bad performance.

Figure A.5: Performance distribution of the six proposed algorithms over the updated instance space, only showing bad performance.

REFERENCES

[1] AL BARRAK GROUP. *Ocean Freight Basics: A Beginner's Guide*. URL: `https://absaco.com/ocean-freight-basics-a-beginners-guide/`. [Online], [CITED JANUARY 2025].

[2] BALDI MM, CRAINIC TG, PERBOLI G & TADEI R. "The generalized bin packing problem". In: *Transportation Research Part E* 48.6 (2012), pp. 1205–1220.

[3] BALDI MM, MANERBA D, PERBOLI G & TADEI R. "Generalized Bin Packing Problem for parcel delivery in last mile logistics". In: *European Journal of Operational Research* 274.3 (2017), pp. 990–999.

[4] BALDI MM, PERBOLI G & TADEI R. "The Stochastic generalized bin packing problem". In: *Discrete Applied Mathematics* 160 (2012), pp. 1291–1297.

[5] BLUM C, SCHMID V & BAUMGARTNER L. "On solving the oriented two-dimensional bin packing problem under free guillotine cutting: Exploiting the power of probabilistic solution construction". In: *ALBCOM Reseach Group* (2012), pp. 1–28.

[6] BOURJOLLY J & REBETEZ V. "An analysis of lower bound procedures for the bin packing problem". In: *Computers and Operations Research* (2003), pp. 395–405.

[7] CACCHIANI V, LORI M, LOCATELLI A & MARTELLO S. "Knapsack Problems - An overview of recent advances. Part I: Single knapsack problems". In: *Computers and Operations Research* 143 (2022), pp. 1–9.

[8] CACCHIANI V, LORI M, LOCATELLI A & MARTELLO S. "Knapsack Problems - An overview of recent advances. Part II: Multiple, multidimensional and quadratic problems". In: *Computers and Operations Research* 143 (2022), pp. 1–14.

[9] CASTRO PM & OLIVEIRA JF. "Scheduling inspired models for two-dimensional packing problems". In: *European Journal of Operational Research* 215 (2011), pp. 45–56.

[10] CHEN C, LEE S & SHEN Q. "An analytical model for the container loading problem". In: *European Journal of Operational Research* 80 (1995), pp. 68–76.

[11] CODES. *Github*. URL: `https://github.com/FN-TL/GBPP.git`.

[12] CRAINIC TG, PERBOLI G, REI W & TADEI R. "Efficient lower bounds and heuristics for the variable cost and size bin packing problem". In: *Computers and Operations Research* 38 (2011), pp. 1474–1482.

[13] CRAINIC TG, FOMENI FD & REI W. “The Multi-Period Variable Cost and Size Bin Packing Problem with Assignment Cost: Efficient Constructive Heuristics”. In: *CIRRELT* 24 (2019), pp. 1–35.

[14] CUNHA T, SOARES C & DE CARVALHO A. “Metalearning and Recommender Systems: A literature review and empirical study on the algorithm selection problem for Collaborative Filtering”. In: *Information Sciences* 423 (2018), pp. 128–144.

[15] DANTZIG GB. “Discrete-Variable Extremum Problems”. In: *Operations Research* 5.2 (1957), pp. 266–288.

[16] DATA INSTANCES. *Bit_bucket*. URL: `https://bitbucket.org/ORGroup/workspace/repositories`.

[17] DETRACK. *Load Planning: Optimizing Delivery Efficiency and Maximizing Profits*. URL: `https://www.detrack.com/blog/load-planning/`. [Online], [CITED JANUARY 2025].

[18] DING R, DENG B & LI W. “Meta-Heuristic Algorithms for the Generalized Extensible Bin Packing Problem With Overload Cost”. In: *IEEE Access* 10.1 (2022), pp. 124858–124872.

[19] DOWSLAND WB. “Two and three dimensional packing problems and solution methods”. In: *Operational Research Society of New Zeland* (1984).

[20] DYCKHOFF TR. “A typology of cutting and packing problems”. In: *European Journal of Operational Research* 44.2 (1990).

[21] EISEMANN K. “The trim problem”. In: *Management Science* 3.3 (1957), pp. 279–284.

[22] ESICUP. *Datasets 2D-rectangular*. URL: `http://www.euro-online.org/websites/esicup/data-sets/`. [Online], [CITED AUGUST 2024].

[23] FENG Y, WANG G, DEB S & ZHAO X. “Solving 0-1 knapsack problem by a novel binary monarch butterfly optimization”. In: *Neural computing applications* 28.7 (2017), pp. 1619–1634.

[24] GILMORE PC & GOMORY RE. “A linear programming approach to the cutting stock problem — Part I”. In: *Operations Research* 9.6 (1961), pp. 849–859.

[25] GILMORE PC & GOMORY RE. “A linear programming approach to the cutting stock problem — Part II”. In: *Operations Research* 11.6 (1963), pp. 863–888.

[26] GUYON I & ELISSEEFF A. “An introduction to variable and feature selection”. In: *Journal of Machine Learning Research* 3 (1974), pp. 1157–1182.

[27] GUIMARES C, ALETI AI& SMITH-MILES KA. “Mapping the effectiveness of Automated test suite generation techniques”. In: *IEEE Transactions and Reliability* 67.3 (2018), pp. 771–785.

[28] GUROBI OPTIMIZATION. *The Leader in Decision Intelligence Technology*. URL: `https://www.gurobi.com/`.

[29] HAOUARI M & SERAIRI M. “Heuristics for the variable sized bin-packing problem”. In: *Computers & Operations Research* 36.10 (2009), pp. 2877–2884.

[30] HOOKER JN. “Needed: An emperical science of algprithms”. In: *Operations Research* 42.2 (1994), pp. 201–212.

[31] HOPPER E & TURTON B. “Problem generators for rectangular packing problems”. In: *Studia Informatica Universalis* 2.1 (2002), pp. 123–136.

[32] JOHNSON DS. “Fast algorithms for bin packing”. In: *Journal of Computer and System Sciences* 8.3 (1974), pp. 272–314.

[33] JOOKEN J, LEYMAN P & CAUSMAECKER P. “Features for the 0-1 knapsack problem based on inclusionwise maximal solutions”. In: *European Journal of Operational Research* 311 (2023), pp. 36–55.

[34] KANG J & PARK S. “Algorithms for the variable sized bin packing problem”. In: *European Journal of Operational Research* 147.2 (2003), pp. 365–372.

[35] KANG Y, HYNDMAN R & SMITH-MILES KA. “Visualizing forecasting algorithm performance using time series instance space”. In: *International Journal of Forecasting* 33.2 (2017), pp. 345–358.

[36] KANTOROVICH LV. “Mathematical methods of organizing and planning production”. In: *Management Science* 6.4 (1960), pp. 366–422.

[37] KLETZANDER L, MUSLIU N & SMITH-MILES KA. “Innstance space analysis for a personnel scheduling problem”. In: *Annals of Methamatics and Artificial Intelligence* 89 (2021), pp. 617–637.

[38] LIU C, SMITH-MILES K, WAUTERS T & COSTA AM. “Instance space analysis for 2D bin packing mathematical models”. In: *European Journal of Operations Resarch* 315 (2024), pp. 484–498.

[39] LIU DS, TAN KC, HWANG CK & HO WK. “On solving multiobjective bin packing problems using evolutionary particle swarm optimization”. In: *European Journal of Operations Resarch* 190.2 (2008), pp. 357–382.

[40] LIU K, SMITH-MILES K & COSTA AM. “Using Instance Space Analysis to Study The Bin Packing Problem”. MA thesis. University of Melbourne, 2020.

[41] MATILDA. *Melborne Algorithm Test Instance Library with Data Analytics*. URL: `https://matilda.unimelb.edu.au/`.

[42] MARTELLO S AND TOTH P. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley-Interscience series in discrete mathematics and optimization. Chichester: Wiley, 1990. ISBN: 0471924202.

[43] PISINGER D & ROPKE S. “A general heuristic for vehicle routing problems”. In: *Computer Operations Research* 34.1 (2005), pp. 2403–2435.

[44] PISINGER D & SIGURD M. “Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem”. In: *INFORMS Journal on Computing* 19 (2007), pp. 36–51.

[45] RAKOTONIRAINY RJ. “Metaheuristic solution of the two-dimensional strip packing problem”. PhD thesis. Stellenbosch University, 2018.

[46] RHENUS LOGISTICS. *Rhenus As Your Partner For Rail Freight*. URL: `https://www.rhenus.group/uz/en/transport/rail-freight/`. [Online], [CITED JANUARY 2025].

[47] RICE JR. “The algorithm selection problem”. In: *Advances in Computers* 15 (1976), pp. 65–118.

[48] RODRIGUES G, CUNHA CB, NETO LG & VIERA JGV. “An adaptive large neighborhood search metaheuristic for the Generalized Bin Packing problem with incompatible categories”. In: *Computers & industrial Engineering* 185 (2023), pp. 1–20.

[49] RSTUDIO. *A language and environment for statistical computing*. URL: `https://www.R-project.org/`.

[50] SCHWERIN P & WASHER G. “The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP”. In: *International Transactions in Operational Research* 4 (1997), pp. 377–389.

[51] SILVA E, OLIVEIRA JF & WASHER G. “2DCPackGen: a problem generator for two-dimensional rectangular cutting and packing problems”. In: *European Journal of Operational Research* 237.3 (2014), pp. 846–856.

[52] SHIPPING SOLUTIONS. *International Air Freight: An Overview*. URL: `https://www.shippingsolutions.com/blog/what-is-air-freight`. [Online], [CITED JANUARY 2025].

[53] SMITH-MILES KA & LOPES L. “Measuring instance difficulty for combinatorial optimization problems”. In: *Computers and Operations Research* 39 (2012), pp. 881–882.

[54] SMITH-MILES KA & MUNOZ MA. “Performance analysis of continuous black-box opptimization algorithms via footprints in instance space”. In: *Evolutionary Computation* 45 (2017), pp. 529–554.

[55] SMITH-MILES KA. “Towards insightful algorithm selection for optimization using meta-learning concepts”. In: *ResearchGate* 1 (2008), pp. 4118–4224.

[56] SMITH-MILES KA, BAATAR D, WREFORD B & LEWIS R. “Towards Objective Measures of Algorithm Performance across Instance Space”. In: *Computers and Operations Research* 45 (2014), pp. 12–24.

[57] SMITH-MILES KA, CHRISTIANSEN J & MUNOZ MA. “Revisiting where are the hard knapsack problems? via Instance Space Analysis”. In: *Computers and Operations Research* 128 (2021), pp. 1–18.

[58] SMITH-MILES KA & TAN L. “Measuring algorithm footprints in instance space”. In: *Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane* 39 (2012), pp. 1–8.

[59] SMITH-MILES KA & BOWLY S. “Generating new test instances by evolving in instance space”. In: *Computers and Operations Research* 63 (2015), pp. 102–113.

[60] SPYDER PYTHON. *The Scientific Python Development Environment*. URL: `https://www.Spyder-IDE.org/`.

[61] STIDSEN TR. “GRASP - A speedy introduction”. In: *DTU-Management / Operations Research* 1 (2012), pp. 1–29.

[62] STIDSEN TR, SORENSEN M & KRISTIANSEN S. “International Timetabling Competition 2011: An Adaptive Large Neighbourhood Search”. In: *DTU-Management / Operations Research* 1 (2012), pp. 1–5.

[63] VEDRAN K. *Supply Chain*. IntechOpen, 2008. ISBN: 953-51-5824-4.

[64] WASCHER G, HAUBNER H & SCHUMANN H. "An improved typology of cutting and packing problems". In: *European Journal of Operational Research* 183 (2007), pp. 1109–1130.