

The Three-Dimensional Container Loading Problem



Xiaozhou Zhao

Southampton Business School

University of Southampton

This thesis is submitted for the degree of

Doctor of Philosophy

January 2017

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Xiaozhou Zhao

January 2017

Acknowledgements

I would like to express my sincere thankfulness to Professor Julia A. Bennell for her guidance and laid-back style of supervision throughout all these years. I also appreciate for the advice and suggestion of Dr. Kathryn Dowsland, Dr. William Dowsland and Professor Tolga Bektaş. I feel grateful towards my family who provides generous financial support. Thank you to all the four legged friends and two legged partners. You have kept me sane.

Abstract

This thesis investigates the three-dimensional container loading problem. We review all literatures published in this area, and explain our unique problem raised from an industry partner. As constructive heuristic remains uncontrollable to us, we design improvement algorithms both to and not to intervene with how constructive heuristic works, namely iterated local search and beam search. New benchmark data sets for multiple containers problem are generated to fill the shortage of challenging data sets. Computational results for homogeneous containers problem indicate that while both approaches work on our problem, beam search remains a favourable choice. We also extend our algorithms to solve heterogeneous containers problem.

To My Past and Future Self

Table of contents

Table of contents	xi
List of figures	xv
List of tables	xvii
1 Introduction	1
1.1 Background	1
1.2 The Problem	3
1.3 Research Aims and Objectives	4
2 A Comparative Review of 3D Container Loading Algorithms	7
2.1 Introduction	7
2.2 Problem Variants	8
2.3 Problem Constraints	10
2.4 Placement Heuristics	12
2.4.1 Wall-building	13
2.4.2 Layer-building	15
2.4.3 Other placement heuristics	16
2.4.4 Sequencing	20
2.5 Improvement Heuristics	22
2.6 Matheuristics	27
2.7 Implementation by Problem Type	29
2.7.1 Single Container Loading	30
2.7.2 The Multiple Container Packing Problem	31
2.8 Experimental Results	32
2.9 Conclusion	42

3	Methodology Review	43
3.1	Introduction	43
3.2	Heuristics and Metaheuristics	43
3.3	Descent Method	44
3.4	Iterated Local Search (ILS)	45
3.5	Genetic Algorithm	47
3.6	Simulated Annealing	48
3.7	Tabu Search	49
3.8	Greedy Randomised Adaptive Search Procedures - GRASP	49
3.9	Variable Neighborhood Search - VNS	50
3.10	Guided Local Search - GLS	50
3.11	Beam Search	50
3.12	Conclusion	52
4	Gower Optimal Algorithms Ltd (GOAL)	55
4.1	Introduction	55
4.2	Gower Optimal Algorithms	56
4.3	Cargo Manager	60
4.3.1	How Cargo Manager Works	60
4.3.2	Constructive Heuristics	62
4.3.3	Challenges in working with Cargo Manager	63
4.4	Investigating Cargo Manager	63
4.4.1	Experimental Data for The Investigation on Cargo Manager's Execution Time	64
4.4.2	New Benchmark Data Generation	66
4.5	Experimental Results	67
4.5.1	Cargo Manager Execution Time	69
4.5.2	Impact of Sequencing	74
4.5.3	Results under Different Constructive Heuristics	77
4.6	Conclusion	80
5	Homogeneous Container Problem	81
5.1	Problem Description	81
5.2	Iterated Local Search with Intelligent Neighborhood (ILSIN)	82
5.2.1	Notation	82
5.2.2	Design and Framework	84
5.2.3	Generate Partial Initial Solution	85

5.2.4	Form a New Neighbourhood	87
5.2.5	Search within the Neighbourhood	89
5.2.6	Kick Phase	92
5.2.7	Generate Final Solution	93
5.3	Iterated Local Search with Descent and Randomness (ILSDR)	93
5.3.1	Notation	93
5.3.2	Design and Framework	94
5.3.3	Generate Partial Initial Solution	94
5.3.4	Form a New Neighbourhood	95
5.3.5	Search within the Neighbourhood	95
5.3.6	Descent Move	95
5.3.7	Generate Final Solution	96
5.4	Iterated Local Search with Simple Neighbourhood (ILSSN)	96
5.4.1	Notation	96
5.4.2	Design and Framework	97
5.4.3	Generate Complete Initial Solution	98
5.4.4	Inter Container Optimisation	98
5.4.5	Intra Container Optimisation	100
5.4.6	Kick	101
5.4.7	Obtain Final Solution	101
5.5	The Proposed Beam Search Algorithm	102
5.6	Computational Results	108
5.6.1	Iterated Local Search Experimental Design	108
5.6.2	Iterated Local Search Results	110
5.6.3	Beam Search Results under Different Parameters	118
5.6.4	Control-Set and Results Comparison	121
5.7	Conclusion	127
6	Heterogeneous Containers Problem	129
6.1	Problem Description	129
6.2	The Revised Beam Search Algorithm	130
6.2.1	Unlimited Heterogeneous Containers Problem	130
6.2.2	Limited Heterogeneous Containers Problem	135
6.3	The Revised Iterated Local Search with Simple Neighbourhood (ILSSN) . .	137
6.3.1	Unlimited Heterogeneous Containers Problem	138
6.3.2	Limited Heterogeneous Containers Problem	140
6.4	Experimental Design	143

6.4.1	Container Types, Costs and Container Numbers	143
6.4.2	Beam Search	144
6.4.3	Iterated Local Search with Simple Neighbourhood	145
6.5	Computational Results	145
6.5.1	Beam Search Results	146
6.5.2	Iterated Local Search with Simple Neighbourhood Results	154
6.5.3	Control-Set and Results Comparison	155
6.6	Conclusion	169
7	Conclusions and Future Work	171
7.1	Conclusions	171
7.2	Future Work	172
	References	175

List of figures

2.1	Examples of Wall- and Layer-building	13
3.1	Pictorial representation of iterated local search	46
3.2	Illustration of a beam search	51
4.1	Cargo Manager	57
4.2	An Example of Cargo Manager's Conservativeness	60
4.3	Input File for Cargo Manager	62
4.4	Generation of a single instance	68
4.5	Execution Time for Different Box Types	71
4.6	Execution Time for Different Box Numbers	72
4.7	Impact of Container Size (25 Box Types, 250 Boxes)	73
4.8	Impact of Container Size (25 Box Types, 500 Boxes)	73
4.9	Impact of Container Size (500 Boxes, 10 Box Types)	73
4.10	Impact of Container Size (500 Boxes, 25 Box Types)	74
5.1	Global Evaluation for Homogeneous Container Problem	104
6.1	Local Evaluation for Unlimited Heterogeneous Containers Problem	131
6.2	Global Evaluation for Unlimited Heterogeneous Containers Problem	132
6.3	Local Evaluation for Limited Heterogeneous Containers Problem	136
6.4	Global Evaluation for Limited Heterogeneous Containers Problem	136
6.5	Unlimited Heterogeneous ILSSN Initial Solution Generation	138
6.6	Limited Heterogeneous ILSSN Initial Solution Generation	142

List of tables

2.1	Sequencing Table	21
2.2	Lists of Papers included in the Results Comparison	36
2.3	Ivancic et al., 1989	37
2.4	Loh and Nee, 1992	38
2.5	Bischoff and Ratcliff (1995) and Davies and Bischoff (1999) with Stability Consideration	39
2.6	Bischoff and Ratcliff (1995) and Davies and Bischoff (1999) without Sta- bility Consideration	40
2.7	Martello et al., 2000	41
4.1	Data Sets for Cargo Manager Investigation	65
4.2	Container Sizes for CMI3	66
4.3	Benchmark Data Sets	67
4.4	List of Sequencing Rules	74
4.5	Results of Different Sequencing Rules for Literature Benchmark Data Sets .	76
4.6	Results of Different Sequencing Rules for Soton Data Sets	77
4.7	Results of Different V Sequencing Rules on BR and IMM	78
4.8	Results of Different V Sequencing Rules on LN and soton	79
4.9	Results of Different Heuristics for Literature Benchmark Data Sets	79
4.10	Results of Different Heuristics for Soton Data Sets	79
5.1	Experimental Design for Iterated Local Search with Simple Neighbourhood	109
5.2	soton1 Results for ILS with Intelligent Neighborhood	111
5.3	soton2 Results for ILS with Intelligent Neighborhood	112
5.4	soton1 Results for ILS with Descent and Randomness	112
5.5	soton2 Results for ILS with Descent and Randomness	113
5.6	Results for ILS with Simple Neighbourhood on data s1c5	114
5.7	Results for ILS with Simple Neighbourhood on data s1c10	115

5.8	Results for ILS with Simple Neighbourhood on data s2c5	116
5.9	Results for ILS with Simple Neighbourhood on data s2c10	117
5.10	soton1 Results Comparison among Initial Solution, ILS with Simple Neighbourhood, ILS with Intelligent Neighborhood, and ILS with Descent and Randomness	118
5.11	soton2 Results Comparison among Initial Solution, ILS with Simple Neighbourhood, ILS with Intelligent Neighborhood, and ILS with Descent and Randomness	119
5.12	Beam Search Results under Different Parameters for s1c5b5	121
5.13	Beam Search Results under Different Parameters for s1c5b10	122
5.14	Beam Search Results under Different Parameters for s1c5b25	122
5.15	soton1 Results Comparison among Initial Solution, Control-Set, ILS with Simple Neighbourhood, and Beam Search	124
5.16	soton2 Results Comparison among Initial Solution, Control-Set, ILS with Simple Neighbourhood, and Beam Search	126
6.1	Costs of Container Types under Different Settings	144
6.2	Beam Search results for Unlimited Heterogeneous Containers Problem on data set soton1	148
6.3	Beam Search results for Unlimited Heterogeneous Containers Problem on data set soton2	149
6.4	Beam Search results for Limited Heterogeneous Containers Problem on data sets s1c5 and s2c5	151
6.5	Beam Search results for Limited Heterogeneous Containers Problem on data sets s1c10 and s2c10	152
6.6	Beam Search Running Time (in seconds) for Unlimited Heterogeneous Containers Problem	153
6.7	Beam Search Running Time (in seconds) for Limited Heterogeneous Containers Problem	153
6.8	Iterated Local Search results for Unlimited Heterogeneous Containers Problem on data set soton1	156
6.9	Iterated Local Search results for Unlimited Heterogeneous Containers Problem on data set soton2	157
6.10	Iterated Local Search results for Limited Heterogeneous Containers Problem on data sets s1c5 and s2c5	158
6.11	Iterated Local Search results for Limited Heterogeneous Containers Problem on data sets s1c10 and s2c10	159

6.12 Results Comparison for Unlimited Heterogeneous Containers Problem under Staged Cost	162
6.13 Results Comparison for Unlimited Heterogeneous Containers Problem under Compromised Cost	163
6.14 Results Comparison for Unlimited Heterogeneous Containers Problem under Reality Cost	164
6.15 Results Comparison for Limited Heterogeneous Containers Problem under Container Setting of 2A 5B 12C	165
6.16 Results Comparison for Limited Heterogeneous Containers Problem under Container Setting of 3A 6B 12C	166
6.17 Results Comparison for Limited Heterogeneous Containers Problem under Container Setting of 4A 10B 24C	167
6.18 Results Comparison for Limited Heterogeneous Containers Problem under Container Setting of 6A 12B 24C	168

Chapter 1

Introduction

The three-dimensional container loading problem is much closer towards our daily life than we might initially realise. It does not matter if you have experience in a shipping yard loading hundreds and thousands of containers, or if you ever see the inside of a container. Almost everything we use in the daily life, from food to clothing, from furniture to automobiles, all at one point were transferred by this simple metal box. It made big savings and speeded up the whole transportation process by packing ‘break bulk’ cargo in uniform containers which could easily be transported between lorry and ship. Indeed, containerisation has transformed global trade with its nearly impossible to be quantified transformative power.

This research focuses on the business applications of the problem. Our target problems are the multiple containers problems, i.e. homogeneous container problem and heterogeneous containers problem. It is an area that has had relative less attention from scholars, especially for heterogeneous containers problem. The following section introduces the field of 3D container loading and our motivation behind carrying out this research. The subsequent section describes the problem tackled in this thesis, and discusses geometric and real-life constraint aspects that need to be considered. The final section states the aims and objectives as well as the structure of this thesis.

1.1 Background

The three-dimensional (3D) packing problem has its most common application in the transportation of goods where goods are packed directly into containers or trucks or first packed

on pallets. Boxes are placed into packing arrangements that will be further packed into containers. While extra containers with low utilisation indicate waste resource, an underestimate of the number of containers may result in left cargo which usually lead to delay in transportation. In addition to the real world constraints such as stability, Load bearing and weight distribution, it makes such packing a complex problem. Majority of literature have their focus on items contained within three-dimensional bins. Almost all the paper satisfy the ‘hard’ geometric constraints with most of them extending their basic models to tackle a selection of real-life constraints. While some papers use terms such as items and bins to outline a more theoretical approach, others use terms boxes and containers instead of items and bins. However, some of these papers still meant to have a theoretical approach of items and bins while they used the terms boxes and containers.

The majority of the literature focus on solving single container problem. The single container problem is an output maximization problem where a subset of boxes are to be packed into a single container where the packed value is to be maximised. A small portion of papers focus on the multiple containers problem, an input minimisation problem where all boxes are to be packed in as few containers as possible. The literature review in chapter 2 contains a full review on the typology.

We are however interested in solving multiple containers problem, i.e. homogeneous container problem and heterogeneous containers problem. In homogeneous container problem, we pack all boxes into as few identical containers as possible. In heterogeneous containers problem, a combination of containers with lowest defined cost is to be found to pack all boxes.

Three reasons motivate our research on the multiple containers problems.

- Multiple containers problem was not the focus of the literature and therefore it was not well tackled.
- Compared to single container problem where maximising utilisation in a single container is the focus of the study, multiple containers problem has a directly larger impact on the industry. Ironically, the cause of it is contributed by the contradictory nature of these two problem types. While a perfectly packed single container may be a success for single container problem, it may cause bad packings in other containers for multiple containers problem and therefore leads to extra container(s) being used. Taken the scale of container shipping within industry, solving multiple containers problem will be more appreciated because it directly tackles the end problem.
- The problem of finding a packing solution with lowest defined cost itself is very inter-

esting. It demands designing improvement heuristic methods which travel along the solution space to reach the optimum. The implementation on an applied problem will help improve our understanding on these methods.

1.2 The Problem

Our research project is sponsored by a software company, Gower Optimal Algorithms Ltd (GOAL). GOAL develops software products providing logistics solutions. For our research project, GOAL provides us their software Cargo Manager (CM) which is capable for packing a single container. The software Cargo Manager is purchased by many companies to be used in daily operation and therefore takes all industry constraints seriously. GOAL requires us to embed CM in our algorithms which will be able to solve multiple Containers problem. Thus it is a guarantee that the solution generated by our algorithms are absolutely industrially feasible.

Before solving multiple containers problem, it is important to identify any constraints and define our objectives. We hereby group constraints into three categories. They are fundamental, literature practical, and real industrial constraints. The three fundamental constraints are considered as ‘hard’ geometric constraints that any packing must be following. The first constraint is that boxes must be placed with their edges parallel to the container walls. Then, all boxes must be placed within the container. At last, intersection between two boxes are not allowed. Any violation of the fundamental constraints is a deal breaker. Beyond these fundamental constraints are those practical constraints which are defined in the literature. Some commonly mentioned ones are box orientation, stability, stacking, weight distribution, weight capacity, multi-drop, prioritization and bearing strength of boxes. Detailed description on each of them is given in chapter 2. These constraints are well-defined and indeed practically important. However, academic models which take one or more of these practical constraints into consideration have not been used in daily operation. The real industrial constraints are the ones our sponsor GOAL specially taking care of for industrial use. Most of the constraints are already mentioned in the literature. However, our sponsor has their restrict standard when implementing them. A discussion in depth is given in chapter 4.

Although multiple containers problem is an input minimisation problem where containers as few as possible are used to pack all available boxes. We set different objectives for homogeneous container problem and heterogeneous containers problem. Although using

minimum number of containers is the objective for homogeneous container problem, we find it too much a broad measurement and may not be sensitive enough when evaluating the solution quality. Therefore, we set overall utilisation across all the used containers as our objective. For heterogeneous containers problem, we assign a unique cost to each container type and our objective is to minimise the total container cost.

1.3 Research Aims and Objectives

The main aim of this research is to use single container constructive heuristic to solve multiple containers problem by building improvement heuristics around the constructive heuristic. In particular, we focus on proposing improvement heuristics with and without problem specific knowledge embedded to enhance the solution quality. We have our research objectives stated below.

1) To review the literature focusing on existing solution methodologies, with the aim of providing insight into some of the critical algorithmic design issues. Besides filling the gap in the literature for an up-to-date review in 3D container loading area, such review of existing literature helps design our own improvement heuristics around a constructive heuristic.

2) To identify specific characteristics of the packing layout generated by the constructive heuristic and design improvement heuristics with different neighbourhood moves for homogeneous container problem. The constructive heuristic we are based on is a commercial product which meets various practical constraints faced in day-to-day operations. Therefore, we use it as a standalone embedment to ensure the feasibility of the solution. Together with the fact that we are only revealed with partial information on how the constructive heuristic works due to privacy reason, the uniqueness of our research provides us the opportunity of exploiting problem specific knowledge based on known characteristics of the constructive heuristic.

3) To implement an approach which allows the constructive heuristic effectively does what it is good at. This approach is contrary to the above approach, and the comparison of their performances will be drawn.

4) To adapt above improvement heuristics for heterogeneous containers problem. Along with a lack of research on this problem type, this research chapter contributes to the day-to-day decision making where different container sizes apply.

The remainder of the thesis is structured as follows. In chapter 2 we draw a comprehensive review on all the 3D container loading algorithms. In chapter 3 we briefly review commonly used improvement heuristics in the area of 3D container loading problem, and explain our decision on the methods we use in our algorithms. Chapter 4 introduces our sponsor Gower Optimal Algorithms Ltd (GOAL) and their software Cargo Manager (CM). We also carry out investigation on CM to draw implementation on building our algorithms. We present our own iterated local search algorithms and beam search approach for homogeneous container problem in chapter 5. Chapter 6 modifies beam search and iterated local search algorithms to solve heterogeneous containers problem. Chapter 7 draws conclusions and layouts future work.

Chapter 2

A Comparative Review of 3D Container Loading Algorithms

2.1 Introduction

The three-dimensional (3D) packing problem is a natural generalization of the classical one- and two-dimensional problems. The most commonly cited application for such problems is the transportation of goods that may be packed directly into containers or trucks or first packed on pallets. With a few exceptions, the literature focuses on items contained within three-dimensional boxes. Even with this restriction, it is clear that the relevance and scope of application is high. Despite its important industrial and commercial applications, historically, publications on this problem area are relatively few in comparison to its one- and two-dimensional counterparts. However, recent years has seen rapid growth in research and publications on this problem, including new research avenues that examine integrating packing with routing and scheduling.

Arranging boxes into a container, truck or pallet is one of the more complex packing problems with respect to real world constraints. While aiming to produce a packing arrangement that makes the best use of resources, an underestimate of the number of containers required to transport certain goods may be very costly in terms of delay or carrying extra containers that are under-utilized, resulting in wasted resource. In addition to the box sizes, there is a host of constraints associated with weight distribution, stacking, stability and support that need to be respected when determining the number of containers. Further, clients may constrain consignments to be carried together, or delivery vehicles need to unload at

several locations. Bortfeldt and Wäscher (2013) provide a comprehensive review of such constraints.

The aim of this paper is to review the literature on 3D container loading. Here, the term *container loading* is used in its broadest sense, in a similar way to Bortfeldt and Wäscher (2013), who recently provided a review that specifically focuses on the inclusion of authors of various constraints encountered in practice but gives relatively little attention to the solution approaches. Our paper is a complementary review to that of Bortfeldt and Wäscher (2013) in that it focuses on the design and implementation of solution methodologies for solving these problems. We also provide an experimental comparison of different algorithms on benchmark data sets to identify the state-of-the-art in solution methods in the area. In order to contain the size of the review, we have excluded pallet loading, irregular objects and do not explicitly review the open dimension problem although some papers addressing this problem are mentioned.

The rest of the paper is structured as follows. The next section provides a detailed description of the problem types reviewed in the paper, following the typology of Wäscher et al. (2007). Section 2.3 briefly discusses the constraints met in practice. Sections 2.4 and 2.5 discuss specific aspects of the solution approaches, these are the placement heuristics (how a layout is constructed) and the improvement heuristics (how to search for better solutions) respectively. Section 2.6 presents the research that examines the use of exact methods. Section 2.7 gives an overview of how the literature is split between different problem variants. Experimental results, along with benchmark data sets are analyzed in section 2.8. Conclusions are given in section 2.9.

2.2 Problem Variants

According to Wäscher et al. (2007), cutting and packing problems can be grouped by dimensionality, assortment of large items, assortment of small items and the objective. Here we are only concerned with three dimensional items that are cuboid. We will call the large items containers and the small items boxes. Boxes may be *identical*, *weakly heterogeneous* (many boxes but a few box types) or *strongly heterogeneous* (few boxes and many box types). If there is more than one container, these can be identical, weakly or strongly heterogeneous. They provide the following problem typology.

If the objective is one of *input minimization*, then the aim is to pack all the boxes in as few containers as possible. Combining the classes of large and small item assortments gives six

unique problems as follows,

- Single Stock-Size Cutting Stock Problem (SSSCSP) if the containers are identical and boxes are weakly heterogeneous,
- Single Bin-Size Bin Packing Problem (SBSBBP) if the containers are identical and the boxes are strongly heterogeneous,
- Multiple Stock-Size Cutting Stock Problem (MSSCSP) if the containers and the boxes are weakly heterogeneous ,
- Multiple Bin-Size Bin Packing Problem (MBSBPP) if the containers are weakly heterogeneous and boxes are strongly heterogeneous,
- Residual Cutting Stock Problem (RCSP) if the containers are strongly heterogeneous and boxes are weakly heterogeneous,
- Residual Bin Packing Problem (RBPP) if the containers and the boxes are strongly heterogeneous,

If the objective is one of *output maximization*, then the aim is to pack a subset of boxes that give the highest value into a fixed set of containers. Here there may be a single container or multiple containers. Seven unique problems can be defined as follows:

- Identical Item Packing Problem (IIPP) if there is a single container and the boxes are identical
- Single Large Object Placement Problem (SLOPP) if there is single container and weakly heterogeneous boxes,
- Single Knapsack Problem (SKP) if there is a single container and the boxes are strongly heterogeneous,
- Multiple Identical Large Object Placement Problem (MILOPP) if there are multiple identical containers and weakly heterogeneous boxes,
- Multiple Heterogeneous Large Object Placement Problem (MHLOPP) if the containers are weakly or strongly heterogeneous and weakly heterogeneous boxes,
- Multiple Identical Knapsack Problem (MIKP) if there are multiple identical containers and strongly heterogeneous boxes,
- Multiple Heterogeneous Knapsack Problem (MHKP) if the containers are weakly or strongly heterogeneous and strongly heterogeneous boxes.

All the above problems assume the containers have fixed dimensions. There is one further case where two dimensions are fixed and one, either the length or height, is variable. Clearly, this is a single container input minimization problem, defined by Wäscher et al. (2007) as the Open Dimension Problem. We do not explicitly review this problem type here, some example papers are Allen et al. (2011), Bortfeldt and Mack (2007), Faina (2000), Fujiyoshi et al. (2009), He et al. (2012), Lai et al. (1998), Li and Cheng (1992), Miyazawa and Wakabayashi (1997, 1999), and Yeung and Tang (2005).

2.3 Problem Constraints

There are three basic constraints that are fundamental to the problem and observed by all research papers. These are stated as follows:

- boxes may only be placed with their edges parallel to the walls of the container,
- all boxes must be placed within the container,
- boxes may not intersect each other.

In many cases only solutions that result in the entire base of the box being fully support are acceptable, others allow a small amount of overhang. In cases where the centre of gravity of a box must be supported, it is assumed to be at the same position as its geometrical centre.

Beyond these basic constraints are a multitude of practical considerations. These are largely associated with box orientation, stability, stacking, weight distribution, weight capacity, multi-drop, prioritization and bearing strength of boxes. These are comprehensively discussed by Bortfeldt and Wäscher (2013), who divide them into constraints in relation to the container, items, cargo, positioning and load. We only briefly describe them here.

There are six unique orientations for a box. Many authors allow boxes to freely rotate, for example, Gehring and Bortfeldt, (1997), Wang et al (2008), Egeblad and Pisinger (2009). Perhaps more realistically is the case when the vertical orientation is fixed but boxes may rotate horizontally, for example, Heassler and Talbot (1990), Chien and Deng (2004). While a few papers (Scheithauer, 1992; Morabito and Arenales, 1994) disallow any rotation among boxes.

Load stability is an important consideration in practice, yet is inconsistently dealt with in the literature. If a load is unstable it can lead to cargo damage and difficulty loading and

unloading. Use of straps and filling gaps with airbags between boxes and between boxes and the container wall is an industry practice, but costly and less desirable. Load stability is most commonly achieved through guaranteeing full support for the bottom of the boxes (Gehring and Bortfeldt, 1997). Others relax this constraint and allow partial support by defining a maximum overhang (Parreño et al., 2008; Parreño et al., 2010). Several papers include further constraints such as requiring at least one side to be attached to another box. Two measurements for evaluating cargo stability are proposed by Bischoff and Ratcliff (1995).

Stacking constraints refer to restrictions on how boxes are placed on top of each other as a result of the bearing strength of the boxes. Each box has a maximum weight per unit of area it can support that may vary depending on the box orientation. Also, Bischoff and Ratcliff (1995) argue that the load bearing strength of a box is primarily provided by its side walls. Therefore, in some cases it would be feasible to place an identical box directly on the top of the existing box, but damage may be caused if a box half the size with half the weight is placed centrally on top. Limits on the height a heavy box can be placed in the container may also be associated with stability.

The weight capacity and how the weight is distributed across the container are both important practical constraints for shipping and handling the already loaded container. The ideal weight distribution is measured if a container's centre of gravity is close to the geometrical mid-point of its floor. An unevenly distributed weight may cause difficulties for certain handling operations. In the case of loading a vehicle, the weight on the axels is the critical measure. Further, containers may not exceed a total weight, and in some scenarios, weight capacity, rather than the limited available loading space, is the binding constraint.

Multi-drop is a situation in which a container will be unloaded in a number of different terminals (Bischoff and Ratcliff, 1995; Lai et al., 1998; Ren et al., 2011). Boxes with different consignments are packed separately as an effective strategy to deal with this situation. Therefore, the arrangement of packing patterns is better to be designed in a way that unloading and re-loading a large part of cargo at every destination is avoided.

The above mentioned constraints, which have direct impact on the packing patterns, have to be taken into consideration within the single container heuristic. Mustafee and Bischoff (2013) try to analyse the trade-offs between loading efficiency and a few above practical considerations by combining placement heuristics with agent-based simulation, while Ramos et al. (2015) focus on the stability constraint and extend it into dynamic stability. Tian et al. (2016) further embed some of the constraints into their work in the form of user preference.

Moreover, Eley (2003) defines two other constraints that impact the distribution of boxes among the different containers. Separation of boxes refers to the situation that boxes of two different types must be stored in separate containers. In practice, foodstuff and perfumery articles are required to be transported separately. Complete shipment refers to the situation that shipment should include all other boxes belonging to a certain sub-set.

2.4 Placement Heuristics

In any heuristic solution approach, there needs to be a mechanism to decide how to put the boxes in the container, whether this is simply to generate an initial solution or as an integral part of the approach. Here we call this the placement heuristic, also commonly known as the construction heuristic. For container loading there are a wide range of approaches. When the mix of boxes are weakly heterogeneous then the most common are wall building and layer building, whereas in the strongly heterogeneous case, boxes will be placed one at a time.

Under wall and layer building schemes, boxes of the same type are arranged in rows or columns to fill one side or the floor of the empty space. A list of empty spaces is created for all the feasible placement positions. Once an empty space accommodates a wall or layer, new spaces are generated. Generally, once a wall or layer is built, the remaining space is treated as a reduced-sized container. Both approaches mimic manual packing by attempting to create flat packing faces.

Along with the decision of how to pack the boxes comes the decision of which box type to pack next. Common approaches include pre-determined ordering and dynamic ordering. The pre-determined ordering is achieved by employing some sort criteria, such as box volume, number of boxes, base area or a certain dimension of a box. Dynamic sorting tends to be based on the usable space remaining after the next placement, volume of remaining boxes of the same type and free floor space.

The following sections discuss the placement heuristics including the sequencing decisions. Many of these papers may employ improvement strategies in addition, these are discussed in the following section.

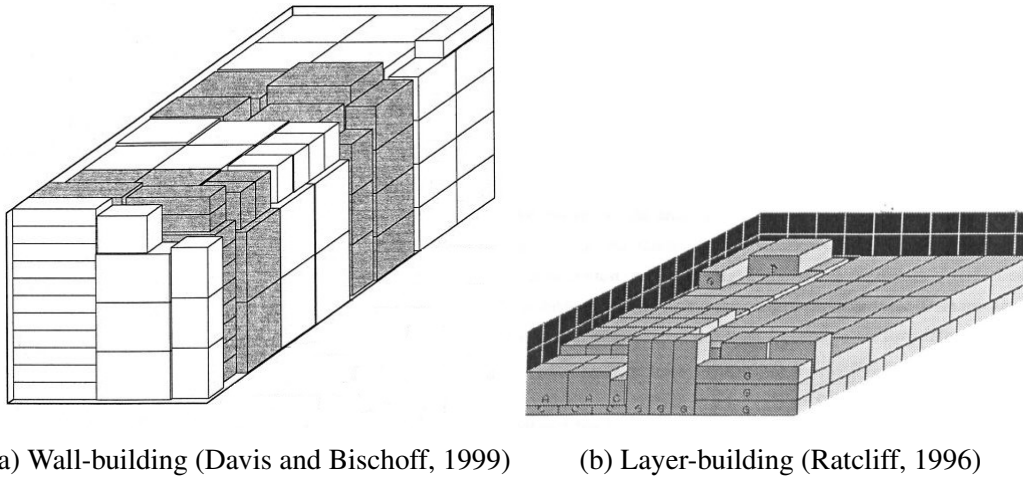


Fig. 2.1 Examples of Wall- and Layer-building

2.4.1 Wall-building

The basic wall-building scheme fills the container with a number of walls across the depth of the container. While the walls are created sequentially, issues around weight distribution can be dealt with by swapping, interchanging or taking the mirror image, provided the boxes do not interlock. While there is no absolute definition of the wall-building placement strategy, the wall-building placement is practically suitable for daily operation where walls consisting of boxes are packed from the back of the container towards the door side. The wall-building placement together with the layer-building placement discussed in section 2.4.2 are the most common placement heuristics one can find in both daily operation and academic literature. Figure 2.1 provides a visual difference of the two different placements with wall-building placement shown in figure 2.1a.

The earliest example of wall-building is by George and Robinson (1980). They design a two-section heuristic based on a real industry example of 800 boxes with no more than 20 types. Given the next empty space, initially the whole container, they select the first box of each wall so that each wall has a size which is not too deep or too shallow using the following ranking criteria. First, select the box whose smallest dimension is the largest among all candidates. In case there is a tie in the first criterion, select the box type with the highest quantity. If there is still a tie, select the box type with the longest largest dimension. Once at least one of a certain box type has been packed, that box type is open. Open box types are given preference where the type with largest quantity has the highest ranking. If the length of unfilled container is too short, usually below a certain defined amount, the rest packing no longer follows the wall-building rules.

Based on the George-Robinson framework, Bischoff and Marriott (1990) compare 14 different heuristics, combined by six ranking rules and three filling methods developed by others, with different ranking criteria. Without a clear winner, they proposed a hybrid version where all 14 heuristics are executed and compared to select the best solution.

Moura and Oliveira (2005) adopt the placement heuristic of George and Robinson (1980) with two modifications. The first modification arises from new space creation. While George and Robinson's heuristic assumes the width of original space as the width of the new created height space, Moura and Oliveira (2005) restrict the space to the area above the packed boxes, citing issue with stability in the former approach. The second modification restricts the width of new wall to be no longer than that of the previous wall, resulting in the amalgamation of unpacked spaces into larger more useful spaces and increasing cargo stability.

Instead of using a fixed sorting criteria, Chien and Wu (1998) and Pisinger (2002) determine the arrangement of boxes by a tree-search algorithm and use the wall approach first proposed by George and Robinson (1980). In Pisinger's (2002) extension of wall building approach, a tree search is designed to find the set of wall depths and strip widths, which can be either vertically or horizontally oriented, to make up the wall. Firstly different wall depths are selected at each branching node, followed by the strip width. Only a fixed number of sub-nodes are considered for each branching node and back tracking is permitted. Several different wall depths are considered according to a total of 27 ranking criteria at each branching node.

Bischoff and Ratcliff (1995) consider the container loading with where cargo has multiple destinations. Their approach packs each consignment in sequence starting with those assigned to the final destination. Given a consignment, the next box packed and its orientation is selected in order to maximize the remaining usable space. In the case of a tie, they select the box space combination with the smallest length protrusion. Two further tie-breakers are to choose the box with largest volume and then to choose the space with shortest width. Here the wall building strategy sequentially adds individual columns or a container width instead of complete walls.

Gehring et al. (1990) pack boxes into a single container of known dimensions using a wall-building approach. The length (thickness) of each wall is decided by the length of the 'Layer Determining Box' (LDB) which is the first, usually largest, box placed in that wall. Two non-overlapping spaces are generated in that section once the LDB is placed. The first space stays next to the LDB with the same height. The second space is above LDB and the first place and is across the container width. Each box is allowed to stay within a wall rather

than project into adjacent walls. Davies and Bischoff (1999) adopt the concept of LDB for generating a block which consists of a fixed number of walls rather than a single vertical wall. Boxes are allowed to bridge the adjacent walls but not two separate blocks. The ranking criteria for selecting box types and orientations are based on Bischoff and Ratcliff (1995). When it is impossible to place another LDB, the current wall is considered as the last wall and its wall depth is extended to the end of the container rather than the length of the LDB. Bortfeldt and Gehring (2001) also use the wall construction heuristic based on LDB as part of a hybrid genetic algorithm.

Bortfeldt and Gehring (1998) and Bortfeldt et al. (2003) construct cuboid configurations of a single box depth, which are effectively walls. In a 1-arrangement, the cuboid consists of as many of one box type as can feasibly fit along the width and height. A 2-arrangement places two 1-arrangement cuboids next to or one on top of the other. All empty spaces are stored in a list, and the space with smallest volume is always packed first. Two available standards are applied alternatively to evaluate the local arrangements within a packing space. The first standard is that the overall volume of the boxes placed in the space should be maximized. The second standard contains double criteria of smallest possible loss volume and largest possible maximum effective volume.

Chien and Deng (2004) describe a container packing support system to determine and visualize, in a step-by-step manner or continuously, the container packing pattern that consists of the packing orientation of each box and corresponding location within a fixed dimensional container. Boxes are packed into vertical strips which later are combined into walls. The spatial matrix representation method is adopted when searching and merging the empty spaces. Liu et al. (2014) and Moon and Nguyen (2014) are two further papers adopting the wall-building strategy in recent research with the latter designing a block-building strategy to improve the wall-building.

2.4.2 Layer-building

Layer building is also a common placement heuristic, although fewer researchers adopt this approach. The packing arrangement is constructed by first placing boxes on the bottom of the container to create the base layer. Once this layer is full, the next layer is constructed on top of the base layer, and so on until no further layers can be contained within the height of the container. An example of layer-building is presented in figure 2.1b in which layers are packed from bottom to top rather than from back to front. It is important to note that some paper describe layers that, according to our terminology, are walls.

The approach of Bischoff and Ratcliff (1995) is inspired by pallet loading. Patterns are built from the container floor upwards in the form of layer building. Each layer contains no more than two different types of boxes, selecting the box type and orientation according to the utilization of the loading surface on which a layer is built. Their motivation for this approach was in tackling stability resulting from significant height differences or gaps exist between adjacent walls of boxes found in other approaches. Lim et al. (2012) follow the same layer building approach as part of an iterative heuristic that focuses on the issue of packing awkward box types. A certain boxes packing priority is revised after each iteration to reflect how “awkward” it was to pack. As a result awkward boxes are packed earlier.

Ratcliff and Bischoff (1998) build on the previous work and design a container loading algorithm which allows for load bearing constraints. Their heuristic approach iteratively packs layers of boxes one at a time from the container floor upwards. Each layer contains no more than two rectangular blocks. Boxes in a single block are of the same types and all have the same orientation. At each step, the algorithm checks the inequality between the weight of the chosen block and the load limit on the loading surface. Boxes with a low load bearing ability can not be packed in the early stages of the procedure. Therefore, the opportunities for future placements on higher layers are examined at each step.

Loh and Nee (1992) propose a layer building construction heuristic. Weakly heterogeneous boxes are grouped according to height and then groups are sorted tallest to shortest. Within each height group, they are sorted by descending base area. Boxes are packed starting at the back of the container, in order, in the first large enough space. Lodi et al. (2002) following a similar sorting strategy, however, group boxes if they are within a certain height tolerance. If the tallest box in the group has height h , then all boxes with height between h and αh are also members of that group. α is set between 0 and 1. Like Loh and Nee (1992), boxes are sorted by non-increasing base area within groups. A two dimensional packing heuristic creates the layer by evaluating and scoring candidate positions. After generating layers that accommodate all the pieces, they solve a one dimensional bin packing problem to pack layer heights within a finite bin height. They call the heuristic Height first - Area second (HA) and combine it with tabu search.

2.4.3 Other placement heuristics

Blocks are homogeneous and each block consists of only identical boxes. Packing container with homogeneous blocks results in several advantages. First of all, it is easy to arrange and requires less loading time. Re-sorting cargo is avoided after unloading as boxes of

same type are stowed closely to one another. The constraint of load bearing strength is less problematic when identical boxes are stacked. The block structure provides extra stability as identical boxes packed in rectangular shape cannot easily slip. Liu et al. (2011a) develop a hybrid tabu search approach in which both a tabu search and a block-building heuristic run iteratively. Box types are sorted in decreasing volume to generate the initial solution. Blocks are made up of the same box type in a configuration to best fit the available space. In some cases another box type is added to improve the fit. Ren et al. (2011) adopt a block placement strategy to tackle shipment priority within a single container. They define five block evaluation functions and use a tree search strategy branching on the best block placement for each evaluation function. Wang et al. (2008) also use a tree structure to represent the container and branch on the mutually exclusive sub-spaces left after placing a block of identical boxes. Zhang et al. (2012) propose a block-building heuristic under which multiple types of boxes can be bound in one single block. A multi-layer search algorithm is designed for block selection. Other block building papers include Alvarez-Valdes et al. (2013a), Wang et al. (2013), Alonso et al. (2014), Elhedhli and Gzara (2015) and Gonzalez et al. (2016). Among all the block building papers, a block could be packed in the form of wall, layer, column or row within a container.

Lai and Chan (1997) propose the maximal space strategy. Maximal space intervals are the set of largest cuboid spaces that entirely cover the unpacked volume of the container. Note these spaces may overlap. After a box is placed, these are generated and stored in a list. Any intervals that are not sufficiently large to contain a box or that are entirely contained within other space intervals are removed from the list. The next box is packed in the closest space to the bottom left corner of the container that is large enough to accommodate the box. Many efficient algorithms adopt the maximal space approach. For example, Parreño et al. (2008; 2010) design a constructive heuristic that can place columns, rows, walls or layers of boxes of the same type. The main idea is to identify maximal spaces and then fill them with the best configuration of identical boxes according to one of two criteria. In each iteration the maximal space with the minimum distance to a corner of the container is filled where space volume is used to break ties. This heuristic is combined with GRASP in Parreño et al. (2008) and Variable Neighborhood Search in Parreño et al. (2010). Other recent papers generating competitive results on benchmark dataset embrace this idea (Gonçalves and Resende, 2012; Zhu and Lim, 2012; Zhu et al., 2012b; Araya and Riff, 2014; Li and Zhang, 2015; Ramos et al., 2016).

Gehring and Bortfeldt (1997) pack boxes into towers or stacks when dealing with strongly heterogeneous boxes. Towers have a base box directly placed on the container floor. Each

of the rest of boxes within that arrangement is placed on another box in a way that its bottom area is entirely supported from below. A greedy algorithm is used to minimize the spaces wasted above the base boxes. In the second step, tower bases are arranged to cover the floor of the container using as a two-dimensional packing problem.

Eley (2002) solves single container loading problem with boxes in block arrangements. The proposed greedy heuristic sorts boxes by volume in the non-increasing order. Each combination of box orientation and empty space is investigated by certain criteria, and the most appropriate empty space is chosen for a box. The main criterion is that after packing a box the sum of the volume of spaces where no remaining boxes can be packed is minimized. In the case of a tie, preference is given to the empty space closest to the lower back left corner of the container. The solutions are further improved through considering different box loading sequences via a tree search where branching is carried out for different box types and box orientations.

Lim et al. (2003) sort the boxes by evaluating a pairwise priority according to the ratio of volume and base area. They propose a new construction approach called multi-faced buildup algorithm where boxes can be placed on any wall in the container as each and every wall can be used as base or floor. Once boxes are placed on the wall of the container to construct the first base, the other boxes will be placed on top of these boxes.

Ngoi et al. (1994) avoid the dependency of the layout on the order of the boxes by evaluating every potential placement location for each box. In order to improve the efficiency of their approach, they use a spatial representation technique. A single 3D matrix (or the spatial matrix), in the form of layers of 2D matrices is used to represent the positions and dimensions of all the packed boxes and empty spaces. Chua et al. (1998) and Chien et al. (2009) also use the spatial representation techniques to model the box packing process and to track both the packed and unpacked space in a single container. Compared to that of Ngoi et al. (1994), their algorithm allows users to pre-assign positions for certain boxes. Spatial matrices are also used by Bischoff (2006) who develops a new construction heuristic to specifically consider load bearing strength. The approach guarantees all boxes are directly placed on the container floor or have their whole base in direct contact with other boxes. The constructive heuristic packs a single box in each iteration and uses a scoring system made up of five criteria to choose between placement alternatives. The available loading surfaces are stored and updated in two matrices, one records the height of surfaces and the other records their corresponding load bearing ability.

The above packing heuristics model available spaces. Martello et al. (2000) and Crainic et al. (2008) both follow an alternative strategy of identifying placement points, the former

calling it corner points, the latter extreme points. These become the candidate placement positions for placing the unpacked boxes. Consider the packing profile over the 3D surface (if overhang not permitted), these points arise from contact between the face of two boxes or a box and the container and are located at the point where three edges coincide to create a fully concave corner. Martello et al. (2000) approach first distributes boxes between bins before determining the specific position of each box. Each bin is constructed using a tree search where each node is a partial solution and there is a child node for each feasible corner point for the next box, where boxes are packed in non-increasing order of volume. Crainic et al. (2008) simultaneously assigns boxes to containers and packs using the well known best fit decreasing heuristic. Zhu et al. (2012c) absorb the idea of both papers and develop the so-called extreme point insertion heuristic.

A similar placement strategy is proposed by Huang and He (2009a) and Huang and He (2009b) to pack boxes into a single container. In their papers they call it caving degree where boxes packed into a corner or even a cave, an empty hollow-like space surrounded by many boxes, whenever possible. The caving degree approach stores all available corners into a list and evaluates all combinations of corners, types of boxes and allowed orientations selecting the best combination of corner, box and orientation based on ranking rules. He and Huang (2010) suggest a modification to the original approach in the effort of decreasing the searching space. He and Huang (2011) make further improvement to the modified approach including adding a local search phase.

Han et al. (1986) construct an L-shape pattern by placing boxes along the base and one vertical edge of the container. George (1992) designs a heuristic to solve the same problem of identical boxes. The heuristic runs iteratively and each iteration creates a layer. The reorientation of the container is allowed and layers can be built against any side wall or even floor rather than only the end wall suggested by common approaches.

Lins et al. (2002) provide a directed graph representation of a packing arrangement that builds on a successful application to the two-dimensional case. Three directed graphs are needed to represent the x , y and z edges of each box within the container. They claim that degeneracy and symmetry lead to a sufficient reduction in problem size to allow for enumeration. Experimental results are for identical boxes with six orientations and a largest problem size of 394 boxes.

Clearly there are many diverse ways of creating arrangements of boxes to load a container in terms of placement rules and static and dynamic ordering. While there is no definitive answer as to the best approach, sorting by volume is a very common characteristic. Chien and Wu (1999) proposed the integration of space allocation and ranking in a decision support

system, although did not provide any experimental evidence.

2.4.4 Sequencing

Sequencing refers to the ordering of the boxes and can be critical to the effectiveness of the packing heuristic. Modifications to the placement heuristics often involve altering the box sequence. Here we summarize the most common box sequences described in the literature. We categorize sequences into two groups: static and dynamic. *Static sequencing* refers to fixed ranking of boxes or box types before packing takes place. *Dynamic sequencing* refers to the criteria that decides the next box or space to be packed during the packing process. Twelve static sequencing rules and five dynamic sequencing rules are identified through the review. Table 2.1 provides a tabulated listing of the sequences found in the literature. The primary ranking criteria for a particular paper is stated as 1, and the first and second tie breaking criteria are indicated as 2, 3, and onwards. It is worth to mention there are two slight exceptions. In Xue and Lai (1997), three criteria are picked and all combinations with a third-tier tie are tried. There is no clear winner of 6 combinations. Among Crainic et al.'s (2008) proposed sequencing rules, two combinations generating best results are remained in the table. However, the first priority of both combinations is slightly different from the ones we defined in this paper. Take the first combination, clustered area and height, as an example. Clustered area means separating base areas into clusters defined by certain intervals. Boxes are then assigned to clusters according to their base areas. Clusters are ordered by certain rule, and boxes within the same cluster are ordered by height.

Year	Article	Static												Dynamic				Placement	Container
		SA	SB	SC	SD	SE	SF	SG	SH	SI	SJ	SK	SL	DA	DB	DC	DD		
1980	George and Robinson, 1980												1					Wall	Single
1990	Gehring et al., 1990																	Wall	Single
1992	Loh and Nee, 1992									2								layer	Single
1993	Lin et al., 1993										1		2					layer	Single
1994	Hemminki, 1994												1					Wall	Single
1995	Bischoff and Ratcliff, 1995												3				2	Wall	Single
1997	Xue and Lai, 1997																	block	Multiple Identical
1998	Bortfeldt and Gehring, 1998																	Layer	Single
1998	Ratcliff and Bischoff, 1998												1					Other Placements	Single
1999	Chien and Wu, 1999												3				2	Wall	Single
1999	Davies and Bischoff, 1999												1					Corner/Extreme Points	Single
2000	Martello et al., 2000												1					Wall	Single
2001	Bortfeldt and Gehring, 2001									3			1					Wall	Single
2002	Gehring and Bortfeldt, 2002												1					Layer	Single
2002	Lodi et al., 2002									1								Block	Single
2002	Eley, 2002												1					Block	Single
2003	Eley, 2003																	Wall	Single
2004	Chien and Deng, 2004									3	4	5					1	Wall	Single
2005	Lim et al., 2005									2			1					Other Placements	Single
2005	Moura and Oliveira, 2005																	Wall	Single
2006	Takahara, 2006									2	3	4	1					Wall	Multiple Identical
2008	Crainic et al., 2008									2								Corner/Extreme Points	Single
2008	Crainic et al., 2008									1								Maximal Space	Single
2008	Pareno et al., 2008												2					Wall	Multiple Multi-sized
2008	Takahara, 2008												1					Multiple Identical	Multiple Identical
2009	Crainic et al., 2009												1					Multiple Identical	Multiple Identical
2010	Almeida and Figueiredo, 2010									3	4	5	1					Block	Single
2010	Fanslau and Bortfeldt, 2010																1	Corner/Extreme Points	Multiple Identical
2010	Wu et al., 2010												1					Wall	Single
2011	Che et al., 2011																	Multiple Identical	Multiple Identical
2011	Derell and Das, 2011									2	1							Wall	Single
2011	Liu et al., 2011												1					Block	Single
2012	Baldi et al., 2012												1					Multiple Multi-sized	Multiple Multi-sized
2012	Kang et al., 2012									1	1	1	1					Other Placements	Single
2012	Zhang et al., 2012																	Block	Single
2012	Zhu et al., 2012c												1					Corner/Extreme Points	Multiple Identical
2013	Alvarez-Valdes et al., 2013a									6	4	5	3					Block (Rows and Columns)	Multiple Multi-sized
2013	Hifi et al., 2013												1					Multiple Identical	Multiple Identical
2014	Baldi et al., 2014												1					Multiple Multi-sized	Multiple Multi-sized
2014	Moon and Nguyen, 2014									1	1	1	1					Wall	Single
2015	Li and Zhang, 2015									1	1	1	1					Maximal Space	Multiple Multi-sized

SA: base dimensions which are the depth and width of the bottom of a box; SB: aggregate base area which is the sum of the base areas of an arrangement; SC: base area of a box; SD: size of the smallest dimension;

SE: the quantity of boxes in the box type; SF: the length of the largest dimension; SG: the depth of a box; SH: the width of a box; SI: the height of a box;

SJ: the perimeter which is the sum of the boxes' 12 dimensions; SK: the weight of a box; SL: the volume of a box;

DA: potential volume utilization or total volume use within the next free space to be packed; DB: the volume of a block which consists of boxes of same type with same orientation; DC: the volume of remaining boxes of the same type;

DD: lengthwise protrusion which is the sum of the coordinate from the packing point and the length of the box to be packed; DE: bottom area of the left free space.

Table 2.1 Sequencing Table

2.5 Improvement Heuristics

While the placement heuristics will provide a fast, and in many cases reasonable quality, solution, broadening the search space using an improvement heuristic can usually provide a significant gain. In general, these work with one or more complete solutions and make neighborhood moves to find better solutions. These vary in sophistication from simple neighborhood structures and acceptance criteria that only accept improving moves, to complex and varying neighborhoods and acceptance criteria that can find many local optima including many implementations of the standard metaheuristics. In this section we review how these techniques have been employed for 3D container loading and how they build on the placement heuristics described in the previous section.

Many of the placement heuristics are dependent on a sorted order, or permutation, of boxes. This representation is often used in a Genetic Algorithm (GA) implementation. Its popularity may be due to the many genetic operators designed to deal with permutation representations in the GA literature.

One of the earliest examples is Hemminki (1994) who uses a GA to determine the widths of the walls. Each chromosome consists of a string of integers. Each integer indicates the packing strategy used in corresponding wall, and all these walls are combined into a whole container. Parent chromosomes are selected randomly and the selection chance of each chromosome is directly linked to its fitness. Crossover or mutation is applied to two parents in order to generate two child chromosomes. Best combinations of walls will be decided through the evolving process of the generations.

The GA of Gehring and Bortfeldt (1997) solves a two-dimensional subproblem. As discussed earlier, boxes are arranged into towers. The tower bases form the two-dimensional packing problem. A chromosome represents the solution by a placement vector, which indicates the sequence of the placements of the tower bases and the two possible orientations of each tower base. Each chromosome is transformed into a solution by identifying an order list of placement corners and placing the boxes in the first feasible placement corner. The population are ranked according to fitness value and the ranking provides the selection probabilities. Since the chromosomes in this cases are permutations of tower bases, a permutation preserving operators are applied to generate descendants representing feasible solutions. Crossover and mutation are alternated randomly. A similar framework for the GA is implemented by Bortfeldt and Gehring (2001). However, in this work the GA is representing layers (walls) of boxes that are constructed using a basic heuristic based on the LDB. The basic heuristic is used to generate starting solutions and new layers for offspring

via crossover and mutation. The chromosome with the best stowage plan finally undergoes resequencing of the layers in order to determine the best weight distribution. Gehring and Bortfeldt (2002) try to improve the solution quality through the application of a parallel genetic algorithm. The new model is implemented in a local PC network and each GA instance is assigned to a separate workstation.

Wu et al. (2010) propose two different approaches to load boxes into a single bin. The second approach is a genetic algorithm. Each bit of the chromosome has two parts, the box and its rotation angle numbered from one to six. Initially all the GA chromosome encode the boxes in non-increasing volume order with a randomly assigned rotation. Reproduction uses roulette wheel selection, a one point crossover with repair mechanisms and a mutation that randomly swaps boxes or changes rotations. The best solution is copied to the next generation.

Gonçalves and Resende (2012, 2013) also use a permutation structure but avoid the need to correct descendent chromosomes by using a biased random key strategy. The strategy considers a permutation of boxes and randomly assign a real number to each box. Sorting the boxes by the random number provides a new permutations. The biased random key approach applies the genetic operator to the random keys rather than the boxes, hence ensuring any offspring is feasible. Their implementation doubles the length of the chromosome to include a placement heuristic for each box.

Dereli and Das (2011) propose an alternative population heuristic that has many similarities to GA but is inspired by foraging behavior of honey bees. The bees algorithm works with a population of solutions and evaluates their fitness, where promising solutions are selected and undergo a neighborhood search phase. The best solutions and some newly generated solution go on to the next population. Layouts are constructed using LDB placement heuristic hence the focus of the search algorithm is to find the alternative widths of layers for a better overall performance through enabling or disabling the rotation of the boxes. Other than getting inspired by honey bees, Silveira et al. (2013) adopt Ant Colony Optimization with the constraint of guillotine cut added to their problem.

Into recent years, GA remains a popular method among academics (Moon and Nguyen, 2014; Bożejko et al., 2015; Zheng et al., 2015; Jamrus and Chien, 2016; Ramos et al., 2016). A few other papers propose evolutionary algorithms similar to GA (Li and Zhang, 2015; Gonzalez et al., 2016; Huang et al., 2016). The majority of the evolutionary algorithms (including GA) solve single container problem, with only a couple of exceptions. Li and Zhang (2015) solve problems with multiple heterogeneous containers, while Wu et al. (2010) only stated that they solve multiple containers problem.

Tabu Search (TS) is one of the most popular metaheuristics in combinatorial optimisation and this is also reflected in how often it has been adopted in 3D packing. Bortfeldt and Gehring (1998) and Bortfeldt et al. (2003) employ tabu search in combination with their wall building heuristic for loading a weakly heterogeneous boxes into a single container. A complete solution is first generated through a basic heuristic. The tabu search algorithm operates on the encoded problem in the form of a packing sequence and fillable packing spaces. Neighborhood moves involve deviating from the fillable space order. The tabu list holds the complete packing sequence of the selected neighbor. Parallelization of the tabu search algorithm is later applied to improve the solution quality (Bortfeldt et al., 2003). The general approach in these papers is built on by Mack et al., (2004), who apply a simulated annealing (SA) algorithm as an alternative to the TS while maintaining the basic heuristic. SA is later combined with TS to form a hybrid search system. Efforts are taken to integrate the advantages of both meta-heuristics and avoid their weakness. Therefore, SA is pre-processed to determine a good starting solution for one or more TS iterations. The parallelization of SA and the hybrid search is introduced in the end. Unlike genetic algorithm, tabu search and simulated annealing are applied widely in both single and multiple containers problem as later summarised in section 2.7.

Liu et al. (2011a) also perform a search over the assignment of boxes to spaces. Instead of altering the free space assigned to a box, the search alters the box packing order using a swap neighborhood. The tabu list length varies with the size of the problem. Liu et al. (2012) is another example of implementing tabu search.

An intuitive way to combine TS and a placement heuristic when solving the multi-container problem is to use the TS to assign boxes to containers and then use the placement heuristic to determine the layout within the container. This is the approach of Jin et al. (2003), who design several neighborhood moves with the aim of eliminating a target bin. Crainic et al. (2009) suggest a two-level tabu search. The first-level tabu search, similar to Jin et al. (2003), focuses on assigning boxes to bins without deciding their placement. The neighborhood focuses on changing bin assignment through swapping and reassigning boxes between containers. The second-level tabu search works to arrange the boxes in each container using an extreme point construction heuristic and searching over the interval graph proposed by Fekete and Schepers (1997, 2004). Each pair of boxes in the same bin will follow a certain approach to form seven feasible combinations.

Instead of assigning boxes to containers Lodi et al. (2002) use tabu search to assign boxes to two dimensional layers. Their unified tabu search approach has been successfully applied in two dimensional bin packing. The tabu search governs the boxes in each layer and attempts

to reduce the total number of layers. The neighborhood moves are designed to empty a specific target layer that is considered the weakest or most empty. A single box from the target is combined with all boxes in k other bins. The neighborhood includes different values of k as well as different boxes from the target layer. The tabu list holds move penalties determined by the outcome of the move.

Guided local search (GLS) has many similarities to TS in terms of using memory to guide the search. Farøe et al. (2003) tackle the SBSBBP, a multiple homogeneous container problem, using guided local search (GLS). The paper is novel in that it permits overlap between boxes within the containers. Given a fixed number of containers, the aim is to find a feasible arrangement of boxes, once found, one container is eliminated and the boxes are distributed over the remaining containers and the search begins again. A neighborhood move arises from moving a single box along one coordinate axes within a single container or moving a box to the same co-ordinate position in another container. Features that guide the local search away from local optima are defined by the volume of overlap between box i and box j and the total volume of i and j . Due to the large neighborhood, fast local search is used to find local minima.

Apart from Mack et al. (2004), SA has not been a common choice of meta-heuristic. Further examples are Lai and Chan (1997) and Faina (2000), both examine the open dimension problem. In the case of Lai and Chan (1997), neighborhood moves arise from swapping two boxes in the packing order and the search is governed by a fairly standard cooling schedule. Results are reported for the 2D problem but not the 3D problem. Faina (2000) also performs neighborhood moves over the sequence of the boxes. A clear disadvantage to this approach is the computational cost of reconstructing solutions after a move. Faina (2000) reports that beyond 64 boxes the computational effort of improving the solution is unfavorable.

While GA, TS and SA are the classic trio of meta-heuristics, Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Search (VNS), are popular predecessors. While GRASP is applied to solve both single container problem and multiple heterogeneous containers problem, the paper implementing VNS only solves single container problem. GRASP is designed to use a greedy solution construction heuristic and perturb characteristics of the construction process by making selections randomly from a restricted candidate list rather than the most greedy choice. Hence, it has clear advantages for combining with a placement heuristic for container loading.

Lai et al. (1998) consider the open dimension problem where boxes are grouped into collections of customer orders and packed as separate sub-lengths of the container. Hence they argue that they can solve each subproblem of minimizing the packing length of each cus-

tomers order. They model the problem as a graph and show that the optimal solution is when the maximum clique is equal to the number of the boxes. They solve the maximum clique problem exactly and the maximal clique problem using GRASP.

Moura and Oliveira (2005) sort the available box types by the volume utilization given the next empty space. The basic greedy strategy chooses the first box type. Under GRASP, they select a random box from a candidate list of the best candidates controlled by a threshold parameter set between zero and one. When set to one the selection is purely random and when set to zero it is purely greedy. The constructed solution is then improved by a first found improve local-search where the neighborhood unpacks the tail of the sequence then repacks greedily forbidding the first box in the tail to be returned to its previous position. A similar implementation of GRASP is used by Parreño et al. (2008) and combined with the maximal-space placement heuristic. This time candidates join the restricted list according to rank (i.e. being in the top $100 \cdot a\%$, where the parameter a is between zero and one) rather than meeting a quality threshold. Initially, a is randomly chosen from a set of possible values. As the search progresses probabilities are attached to each possible value according to its past success. A couple of other GRASP implementations are Alvarez-Valdes et al. (2013a) and Alonso et al. (2014).

Following the successful implementation of GRASP, the same authors combine their maximal-space heuristic with VNS algorithm (Parreño et al., 2010). The authors propose five types of neighborhood move for the descent phase of the VNS and an additional neighborhood move for the shaking phase. These are layer reduction that removes rows or columns from a layer and attempts to fill the empty space with other boxes, column insertion adds a new column in one of the empty spaces, box insertion is similar to column insertion but only adds a single new box, and the last two moves are to empty an already packed region and to fill it using best-volume for one neighborhood and best-fit for the other. The shaking neighborhood eliminates between 10% and 30% of the boxes in the current solution and refilling with the best-volume criterion.

While the meta-heuristic approaches described above seek to be less greedy or at least include phases that allow a more diverse search of the solution space, other authors simply perform a greedy local search many times. For example Takahara (2006, 2008) perform local search over the permutation of pieces and piece orientation via swap moves. Multiple starting solutions provide a broader search of the solution space. Another approach is to make the greedy choice after looking ahead a few steps, thus reducing the short-term focus of the decision. Local search is also used by Trivella and Pisinger (2016) and Zhu et al. (2012c) where the latter implemented a simple iterated local search strategy called bin

shuffling. So far the focus of all papers applying local search method is only on multiple homogeneous container problem.

Zhu and Lim (2012) describe a look-ahead tree search that decides the next placement on the basis of the best node of an expanded tree of depth d where the m best child nodes are expanded at each node. A predecessor of this paper, Eley (2002) uses a restricted tree search to construct a solution. The tree is searched breadth first and a limit is imposed on the number of child nodes explored. In addition, to avoid a dominant solution path, if two nodes at the same depth have the same evaluation function and the item is packed, then one of the nodes is removed. They call the tree search the pilot method. The tree search method is also applied in Liu et al. (2014). Furthermore, beam search, a tree search alike technique, is also adapted by academics (Wang et al., 2013; Araya and Riff, 2014; Baldi et al., 2014). Tree search method is applied widely to solve all three types of problems, i.e. single, multiple homogeneous and multiple heterogeneous containers problem.

2.6 Matheuristics

Compared to the number of heuristics available, the number of exact algorithms in 3D container loading are limited. One reason behind this limitation is the difficulty in representing possible patterns or practical packing constraints. Even if this can be done (and we provide pointers to the literature below where it is), another difficulty is the solution of the resulting formulation, which is often large-scale due to the number of containers and boxes. Nevertheless, encouraging progress has been made along this line of research for which a review is presented below.

Mohanty et al. (1994) describe an algorithm for the (3D) BPP, where the aim is to pack boxes of different types, each with a prespecified number, length, width and height, into a set of containers of different sizes, with the objective of maximizing the total value of packed boxes. The problem is originally formulated as a multidimensional knapsack problem, including an integer variable that is defined with respect to each container type packed with respect to a given pattern. As the number of such patterns is excessively large, the complexity of solving the formulation to optimality is a challenge. For this reason, the authors describe a column generation procedure where the pricing problem involves solving an integer program, in the form of a general fractional knapsack problem with side constraints, which itself is difficult to solve. Instead, the authors propose heuristics to solve it. Although the overall solution approach is heuristic in nature, it is interesting that elements of exact

solution methods (i.e. mathematical formulations) are integrated within the approach. In a relevant study, Eley (2003) looks at a similar problem setting, possibly with some side constraints, and an objective function that minimizes the number of bins used using a two-stage column generation algorithm. In the first stage, a limited number of patterns (columns) are generated *a priori* using a heuristic, which are then fed into an integer program that is solved to optimality in the second stage. The algorithm has the advantage of being able to incorporate a number of practical constraints, such as box orientation or load stability. The computational results that the author presents on benchmark problems suggests that the integer program does not require more than five seconds of solution time, and the algorithm itself outperforms the integer programming based heuristic approach purposed by Ivancic et al. (1989) and the heuristic method by Bortfeldt (2000).

Given the success of column generation, it is not surprising to see that Zhu et al. (2012a) have also described a similar algorithm for the Multiple Bin-Size BPP, where the aim is to minimize the cost of containers used. The problem also incorporates some side constraints, namely full-support and partial-support for boxes and that all boxes are packed orthogonally. The pricing problem here is a single container loading problem. Instead of generating the actual columns, which is costly, the authors instead resort to a strategy where a reasonable approximation of the columns, named prototype columns, are generated. Compared with the algorithm of Che et al. (2011a), the proposed method finds solutions that improve the average gap from the optimal solutions by about 50%. The recent work of Wei et al. (2015) also has column generation embedded in their algorithm.

An alternative approach to mathematical models where variables are defined with respect to each possible pattern (or packing) has been to use formulations where variables are defined with respect to placement of individual items. Such an approach avoids the problem of dealing with a large set of patterns, and instead relies on a discretization of the container space and an explicit representation of decisions on whether items are placed in certain coordinates or not. Chen et al. (1995) present such a model in the form of a mixed integer linear programming formulation for the general 3D container loading problem, with an objective to minimize the unused space of the containers used. The model is used to solve a sample instance of three heterogeneous containers and six boxes using LINGO, with a solution time of 15 minutes. The authors also report results on another sample instances with a single container and six boxes, this time to minimize the required length of the container. A polyhedral study of an improved version of this formulation is presented by Padberg (2000). The study provides some facet results and valid inequalities, with the recommendation that they are used within a branch-and-cut algorithm to solve the problem. Allen et al. (2012)

provide empirical evidence that the computational reach of this formulation is only about 20 boxes using modern solvers. The authors present an alternative formulation that is based on space-index variables, which has better performance than that of the Chen et al. (1995) and Padberg (2000), both with respect to the size of the instance and the computational time required to solve to optimality. Wu et al. (2010) build on the work of Chen et al. (1995) for the open dimension problem.

For a mixed integer programming formulation of the BPP with identical bins, see Hifi et al. (2010), who also present some valid inequalities and computational results. Another model based on the Cartesian coordinate system is presented by Junqueira et al. (2012a) for the 3D container loading problem, where constraints representing vertical and horizontal stability and load bearing strength of the boxes are explicitly represented in the formulation. Computational results presented by the authors show that by using CPLEX 11.0 with default parameters, the formulation is able to solve randomly generated instances with four types and up to 20 boxes, and between 10 and 100 containers, in most cases to optimality. The difficulty of finding an optimal solution increases with the number containers as well as with the similarity of dimensions of the boxes. An extension of this model to the variation of the problem with multi-drop constraints is presented in Junqueira et al. (2012b).

Lim et al. (2013) present a heuristic algorithm for a problem named the single container 3D loading problem with axle-weight constraints, which arises due to the weight restriction for trucks which carry containers when they are offloaded from ships. This additional restriction introduces nonlinearities in a possible mathematical programming formulation of the problem. The authors describe a heuristic procedure, but make use of integer linear programming formulations, either mixed or binary, within the algorithm to improve the packing.

Finally, we mention the work by Hifi (2002) who presents approximation algorithms for container loading problems. The algorithms are based on the idea of solving a series of knapsack problems to form stacks, and then putting the stacks into the container by solving unbounded knapsack problems to optimality.

2.7 Implementation by Problem Type

In this section we categorise the papers according to the problem types they address. Bortfeldt and Wäscher (2013) provide a useful table that identifies each paper by its problem type. Here we go further and identify the features of the methodologies employed for each

of these problems.

2.7.1 Single Container Loading

Among the papers we have reviewed, twenty-six tackle SLOPP. The type and sophistication of the methodologies vary greatly. Thirteen of the paper only employ a placement heuristic, and of these thirteen seven use a static ranking and a placement criteria (Bischoff and Marriott, 1990; Chien and Deng, 2004; Chien et al., 2009; Christensen and Rousøe, 2009; George and Robinson, 1980; Loh and Nee, 1992; Ren et al., 2011). The remaining six use dynamic selection of the next box or space (Bischoff and Ratcliff, 1995; Davies and Bischoff, 1999; Ratcliff and Bischoff, 1998; Wang et al., 2008), or a tree search to make the exploration less greedy (Lins et al., 2002; Morabito and Arenales, 1994). Eleven papers augment the placement heuristic with an improvement heuristic, or design an improvement heuristic that directly tackles the problem. Of these two use genetic algorithm (Gonçalves and Resende, 2012; Kang et al., 2012), three use tabu search (Bortfeldt et al., 2003; Liu et al., 2012a; Mack et al., 2004) and one uses simulated annealing (Mack et al., 2004), with the rest adopting various local search techniques (Bischoff, 2006; Burke et al., 2012; Eley, 2002; Hifi, 2002; Moura and Oliveira, 2005; Parreño et al., 2008, 2010). Junqueira et al. (2012b) is the only paper we reviewed that develops exact methods for SLOPP, while Gonzalez et al. (2016) draw a multi-objective formulation of volume and weight with the placement constraint of guillotine cuts.

Thirty-three papers aim to solve SKP. The methodologies differ largely in terms of type and sophistication. Ten of the paper only employ a placement heuristic, and of these ten five use a static ranking and a placement criteria (Chien and Wu, 1999; Gehring et al., 1990; Haessler and Talbot, 1990; Lim et al., 2003; Liu et al., 2011b). The remaining five use dynamic selection of the next box or space (Burke et al., 2012; Lim and Zhang, 2005; Ngoi et al., 1994; Scheithauer, 1992), or a tree search to make the exploration less greedy (Chien and Wu, 1998). Nineteen papers augment the placement heuristic with an improvement heuristic, or design an improvement heuristic that directly tackles the problem. Of these nineteen eight use genetic algorithm (Bortfeldt and Gehring, 2001; Gehring and Bortfeldt, 1997, 2002; Gonçalves and Resende, 2012; Hasni and Sabri, 2013; Liang et al., 2007; Lin et al., 1993; Yeh et al., 2003), one uses tabu search (Liu et al., 2011a) and one uses simulated annealing (Egeblad and Pisinger, 2009), with the rest adopting various local search techniques (Fanslau and Bortfeldt, 2010; He and Huang, 2011, 2012; Huang and He, 2009a, 2009b; Lim et al., 2005; Parreño et al., 2008, 2010; Pisinger, 2002). Finally the following

four papers develop exact methods for SKP, Fekete et al. (2007), Junqueira et al. (2012a, 2012b), Padberg (2002).

Besides all the single-container papers mentioned above, a few exceptions are identified. Ramos et al. (2016) design their algorithm for both SLOPP and SKP problems. While Liu et al. (2012), Moon and Nguyen (2014) and Wang et al. (2013) do not declare which problem type their algorithms are designed for, they run computational experiments using data sets with only weakly heterogeneous boxes. On the other hand, Alonso et al. (2014) and Liu et al. (2014) use data sets with both weakly and strongly heterogeneous boxes in their computational experiments without declaring which problem they specifically intend to address. Some other papers only clarify their intention for solving single container problem without mentioning whether weakly or strongly heterogeneous boxes are involved (Bożejko et al., 2015; Huang et al., 2016; Jamrus and Chien, 2016; Zheng et al., 2015).

2.7.2 The Multiple Container Packing Problem

To load multiple containers, three strategies are usually suggested: sequential strategy, pre-assignment strategy and simultaneous strategy (Eley, 2003). Under a sequential strategy (de Castro Silva et al., 2003; Thapatsuwan et al., 2012; Wu et al., 2010; Xue and Lai, 1997), containers are filled one after the other. A new container is opened once the current containers can not store any of the available boxes. However, one of the disadvantage of this strategy is that containers filled last are at risk of poor volume utilization ratios since large or awkwardly formed boxes are often left to the end. The pre-assignment strategy first distributes boxes among different containers. No actual packing occurs at the distribution stage. A heuristic is then applied to load assigned boxes into each container. Boxes not be packed in their designated containers are re-packed to other containers under additional mechanisms. Jin et al. (2003) is a typical example of adopting this strategy. The simultaneous strategy adopted by Chen et al. (1995), de Almeida and Figueiredo (2010), and Soak et al. (2008) attempts to assign and pack boxes into more than one container at a time. It is surprising to learn that the sequential strategy outperformed the simultaneous strategy in Eley's 2002 approach. A few exceptions are Che et al. (2011a), Crainic et al. (2009) and Farøe et al. (2003) who all apply mixed strategies. Che et al. (2011a) firstly pack boxes into packing patterns and then load one container at a time based on these patterns. Crainic et al. (2009) build initial solution under a sequential strategy but apply improvement phase using pre-assignment strategy. Farøe et al. (2003) construct an initial solution by firstly generating a number of walls before combining them into whole bins.

Seven papers address SSSCSP. Among them, Bischoff and Ratcliff (1995) and Ivancic et al. (1989) only propose placement heuristic, while Che et al. (2011a, 2011b), Eley (2002) and Kang et al. (2010) design improvement heuristic. Sciomachen and Tanfani (2007) is an exceptional paper which determines container stowage plans in a ship by comparing its own problem to SSSCSP in 3D packing. We reviewed sixteen SBSBPP papers. Most paper in this category only focus on the construction of placement heuristic (Amossen and Pisinger, 2010; Burke et al., 2012; Crainic et al., 2008; de Castro Silva et al., 2003; Epstein and Levy, 2010; Hifi et al., 2013; Lim and Zhang, 2005; Martello et al., 2000; Miyazawa and Wakabayashi, 2009). Among those proposed improvement heuristic, tabu search is surprisingly chosen by most papers (Crainic et al., 2009; Jin et al., 2003; Lodi et al., 2002, 2004), while Farøe et al. (2003) propose a guided local search and Li and Zhang (2015) design evolutionary algorithms. Hifi et al. (2010) is the only paper to use exact methods although these are embedded within a broader heuristic, while Boschetti (2004) suggests a new lower bound. Among MSSCSP papers, Ivancic et al. (1989) only give a placement heuristic while others (Brunetta and Gregoire, 2005; Che et al., 2011a, 2011b; Eley, 2003) all design improvement heuristic. Five MBSBPP papers (Alvarez-Valdes et al., 2013a; Brunetta and Gregoire, 2005; Ceschia and Schaerf, 2011; de Almeida and Figueiredo, 2010; Jin et al., 2003) contain improvement heuristic, while Alvarez-Valdes et al. (2013b) works on the lower bounds. Chen et al. (1995) claim their paper can solve all problem types including RCSP. However, no algorithm is proposed to tackle specifically RCSP. Two papers (de Almeida and Figueiredo, 2010; Jin et al., 2003) in our collection solve RBPP. Similar to RCSP, Hifi's (2002) algorithm is capable of solving MILOPP but it is a problem type remaining almost untackled. MIKP is the third problem type remaining unexplored. Eley (2003) and Mohanty et al. (1994) are the only two papers we found solving MHLOPP. The paper of Ceschia and Schaerf (2011) also solves MHKP. While Elhedhli and Gzara (2015) only solve multiple identical containers problems, Baldi et al. (2012; 2014) and Wei et al. (2015) claim that they intend to solve problems with multi-sized containers.

2.8 Experimental Results

One aim of this paper is to provide a comparison of the various algorithms described above based on their performance on benchmark data sets, in an attempt to identify the state-of-the-art algorithms for this class of problems. To our knowledge, five classes of benchmark data sets exist for 3D loading problems. These are:

1. Ivancic et al. (1989): This data set contains 47 instances of weakly heterogeneous

- boxes. Only one container size is assigned to each instance, but different container sizes are assigned across different instances. The number of containers should be kept as small as possible as is typical in the Single Stock-Size Cutting Stock Problem (SSSCSP).
2. Loh and Nee (1992): This data set contains weakly heterogeneous boxes and a single container in each of its 15 problem instances. Boxes may be left over if the single container is full. This dataset is designed for the Single Large Object Placement Problem (SLOPP).
 3. Bischoff and Ratcliff (1995): This set is for the Single Large Object Placement Problem (SLOPP) which packs weakly heterogeneous boxes into a single container. There are seven classes in total with each class including 100 instances.
 4. Davies and Bischoff (1999): This set pertains to the Single Knapsack Problem (SKP) which packs strongly heterogeneous boxes into a single container. There are eight classes in total with each class including 100 problem instances. This set is usually merged with Bischoff and Ratcliff (1995) to become a so-called BR1-15 dataset.
 5. Martello et al. (2002): This set is for a typical Single Bin-Size Bin Packing Problem (SBSBPP) which packs strongly heterogeneous boxes into a minimum number of identical containers. There are eight classes of problem that are randomly generated according to certain parameters, but the second and the third classes are not included in the result tables due to their similarity to the first class.

In the rest of the section we provide comparisons of all algorithms published in papers listed in Table 2.2, which also presents the running environment of each algorithm. Then, we present comparison results for each data set. In particular, Table 2.3 is for the data set by Ivancic et al. (1989) and Table 2.4 is for the data set by Loh and Nee (1992). Tables 2.5 and 2.6 jointly present the data sets by Bischoff and Ratcliff (1995) and Davies and Bischoff (1999). Finally, Table 2.7 presents the comparison results for the instances by Martello et al. (2002).

Table 2.2 lists all the papers whose algorithms we compare in this section. Papers are listed by year of publication and we assign a code that contains the first letter of the author's name and year of publication. In cases where more than one method is presented in the same paper, extra letters, usually short for the methods (and often assigned by the authors themselves), are added to the end. For example, two methods are presented in Lodi et al. (2002). We therefore include both methods in the table under names of L2002-HA and L2002-TS. Some authors present two sets of results, one that takes account of box stability

(S), and another where stability is ignored (U). In these cases we add letter S for when box stability is included, and letter U when it is not. An example would be Fanslau and Bortfeldt (2010) as we differentiate them as FB2010S and FB2010U. Gehring and Bortfeldt (1997), as well as Bortfeldt and Gehring (1998), re-ran their algorithm after the publication of their original papers using a more powerful PC. In this case the new codes GB1997N and BG1998N are assigned. The last two algorithms are separated in the table because they are lower bound generators.

In Table 2.3 the number of box types and total number of boxes are provided for each of the 47 instances. For the 13 methods compared, results are presented in term of the number of containers used. In some cases average running times are reported. A lower bound is also presented in the last column of the table. Among all the methodologies, the column generation combined with heuristics approach of Zhu et al. (2012a) generates the best results when box stability is not considered. However, it has a much higher average running time compared to other methods.

Table 2.4 contains 15 problem instances and the number of boxes is given for each instance. Overall, the results of 17 algorithms are presented. While 16 results are shown by the percentage utilisation rate in the columns of U%, the original Loh and Nee (1992) paper uses packing density (D%) to present their results. Instead of using the original three container dimensions, the maximum used container dimensions are chosen. The number of boxes leftover (BL), running time in seconds indicated as rt(s) and stopping time in seconds are presented for each method, if they are reported by the original authors. The best result of a 71% overall average utilisation rate is generated by He and Huang (2011). They claim that their heuristic method is inspired by an old Chinese proverb together with a fit degree algorithm (FDA) their algorithm is designed to solve weakly heterogeneous problems.

Data set Bischoff and Ratcliff (1995) (reported as BR1-7) and Davies and Bischoff (1999) (generally reported as BR8-15) are combined making it a total of 15 data sets, namely BR1-15. Tables 2.5 only includes those results taking stability into consideration while Tables 2.6 consists of results without taking stability into account. Numbers of box types are given to each class for both tables. The results are presented as percentage utilisation. Running times and stopping time are also presented if they are available. The beam search based algorithm of Araya and Riff (2014) outperforms all other methods in both tables, except for data sets BR8-BR15, on which the approach of Ramos et al. (2016) with stability consideration performs very well. Also, in Table 2.5 when stability is taken into consideration their approach uses considerably less running time than others. However, it is worth pointing out that in Table 2.5 although all algorithms attempt to take stability into account, the levels of stability

are still different and difficult to identify. While some algorithm have the base of all boxes fully supported, others permit a certain percentage of the base to be supported. Therefore, inferior results don't necessarily mean the algorithm is less competitive.

Bin dimension and number of items are presented for each class in Table 2.7. Results are in the form of the number of bins used, and stopping time is presented if available. A lower bound is also presented in the last column of this table. The two-level tabu search proposed by Crainic et al. (2009) outperforms other methods. To sum up, all tables in this section show clear improvements on results along the time. Partly benefiting from the development of computer technology, it proves that more competitive algorithms are being developed through all these years.

Year	Paper	Code	Running Environment
1989	Ivancic et al., 1989	IMM1989	IBM PC System 2, model 50
1990	Bischoff and Marriott, 1990	BM1990	(not stated)
1992	Loh and Nee, 1992	LN1992	HP9000 Series 300 workstation with the CPU rated at 5 MIPS
1994	Ngoi et al., 1994	N1994	80386 processor with 33 MHz clock speed
1995	Bischoff and Ratcliff, 1995	BR1995a	(not stated)
1995	Bischoff and Ratcliff, 1995	BR1995b	(not stated)
1995	Bischoff and Ratcliff, 1995	BR1995C	(not stated)
1997	Gehring and Bortfeldt, 1997	GB1997	Pentium/130 MHz PC
1997	Gehring and Bortfeldt, 1997	GB1997N	Pentium PC with a frequency of 400 MHz
1998	Bortfeldt and Gehring, 1998	BG1998	Pentium PC with a frequency of 200 MHz
1998	Bortfeldt and Gehring, 1998	BG1998N	Pentium PC with a frequency of 400 MHz
1998	Chua et al., 1998	C1998	DOS platform; IBM PC
2000	Bortfeldt, 2000	B2000	Pentium PC (200 MHz)
2000	Martello et al., 2000	MPV2000	HP9000/C160 160 MHz
2000	Martello et al., 2000	MPV2000-BS	HP9000/C160 160 MHz
2000	Martello et al., 2000	MPV2000-SPack	HP9000/C160 160 MHz
2001	Bortfeldt and Gehring, 2001	BG2001	Pentium PC with a frequency of 400 MHz
2002	Eley, 2002	E2002-seq	Pentium C with 200 MHz clock and 32MB memory
2002	Eley, 2002	E2002-sim	Pentium C with 200 MHz clock and 32MB memory
2002	Eley, 2002	E2002	Pentium C with 200 MHz clock and 32MB memory
2002	Gehring and Bortfeldt, 2002	GB2002	slave on Pentium PC, 400 MHz; master on 486 PC, 33 MHz
2002	Lodi et al., 2002	L2002-HA	Digital 500 workstation with a 500 MHz CPU
2002	Lodi et al., 2002	L2002-TS	Digital Alpha 533MHz
2003	Bortfeldt et al., 2003	B2003	Pentium with a frequency of 2 GHz
2003	Bortfeldt et al., 2003	B2003PS	Pentium with a frequency of 2 GHz
2003	Eley, 2003	E2003	Pentium II PC with a 266 MHz clock and 192MB memory
2003	Farøe et al., 2003	F2003	Digital 500 workstation with a 500 MHz CPU
2003	Lim et al., 2003	L2003	alpha 600 MHz machine
2004	Mack et al., 2004	M2004	several Intel Pentium computers with a frequency of 2GHz combined in a LAN
2005	Lim et al., 2005	L2005	(not stated)
2005	Lim and Zhang, 2005	LZ2005	2.40 GHz Pentium 4 PC with 128 MB memory limit
2005	Moura and Oliveira, 2005	MO2005	Pentium IV at 2.4GHz with 480Mb RAM
2006	Bischoff, 2006	B2006	1.7 GHz Pentium 4 computer with 256 Mb of RAM
2006	Takahara, 2006	T2006	Pentium4 PC with 2.8GHz and 1GB memory
2008	Takahara, 2008	T2008	Xeon PC with 3.0GHz and 3GB memory
2008	Parreño et al., 2008	P2008	Pentium Mobile at 1,500MHz with 512MB RAM
2008	Wang et al., 2008	W2008	Pentium PC
2008	Crainic et al., 2008	C2008	Pentium4 2000 MHz CPU
2009	Crainic et al., 2009	C2009	Pentium4 2000 Mhz CPU
2010	Fanslau and Bortfeldt, 2010	FB2010S	AMD Athlon processor (64 FX60, 2.6GHz) with 2,048MB RAM
2010	Fanslau and Bortfeldt, 2010	FB2010U	AMD Athlon processor (64 FX60, 2.6GHz) with 2,048MB RAM
2010	He and Huang, 2010	HH2010	Intel(R) Xeon(R) 2.33 GHz and 1 GB RAM
2010	Parreño et al., 2010	P2010	Pentium Mobile @ 1,500MHz with 512MB RAM
2011	Che et al., 2011a	C2011	Intel Xeon E5430 CPU @ 2.66 GHz, 8GB RAM
2011	Derele and Das, 2011	DD2011	(not stated)
2011	He and Huang, 2011	HH2011	Intel(R) Xeon(R) 2.33 GHz
2011	Liu et al., 2011a	L2011	Intel Centrino Duo CPU 1.66GHz and RAM 1GB
2011	Ren et al., 2011	R2011	Intel Core 2 U9300 PC (1.2GHz, 2GB RAM)
2012	Gonçalves and Resende, 2012	GR2012S	AMD 2.2 GHz Opteron 6-core CPU
2012	Gonçalves and Resende, 2012	GR2012U	AMD 2.2 GHz Opteron 6-core CPU
2012	Lim et al., 2012	L2012	2.40GHz Pentium 4 pc with 512MB memory limit
2012	Liu et al., 2012	LIU2012	(not stated)
2012	Zhang et al., 2012	Z2012S	Intel(R) Xeon(R) X5460 @ 3.16GHz
2012	Zhang et al., 2012	Z2012U	Intel(R) Xeon(R) X5460 @ 3.16GHz
2012	Zhu and Lim, 2012	ZL2012S	Intel Xeon E5520 Quad-Core CPUs @ 2.27GHz with 8G RAM
2012	Zhu and Lim, 2012	ZL2012U	Intel Xeon E5520 Quad-Core CPUs @ 2.27GHz with 8G RAM
2012	Zhu et al., 2012a	ZE2012aS	Intel Xeon E5520 CPU @ 2.26GHz, 8GB RAM
2012	Zhu et al., 2012a	ZE2012aU	Intel Xeon E5520 CPU @ 2.26GHz, 8GB RAM
2012	Zhu et al., 2012b	ZE2012b	Intel Xeon E5520 Quad-Core CPUs @ 2.27 GHz
2013	Hifi et al., 2013	H2013-EHGH2	3.3 GHz processor with 2 GB RAM
2013	Hifi et al., 2013	H2013-Ampv	3.3 GHz processor with 2 GB RAM
2014	Alonso et al., 2014	A2014	Intel Core Duo T6500 with 2.1 Ghz and 4 GB of RAM
2014	Araya and Riff, 2014	AR2014S	Intel Xeon @ 2.20GHz and 8GB RAM with 2 quad-processors
2014	Araya and Riff, 2014	AR2014U	Intel Xeon @ 2.20GHz and 8GB RAM with 2 quad-processors
2014	Liu et al., 2014	L2014	Core2 Q8300@2.5 GHz
2014	Moon and Nguyen, 2014	MN2014	Intel Core 2 Quad 2.4 GHz processor with 2 GB RAM
2016	Ramos et al., 2016	R2016S	2 Intel Xeon CPU E5-2687W at 3.1 Ghz with 128 Gigabytes of RAM
2016	Ramos et al., 2016	R2016U	2 Intel Xeon CPU E5-2687W at 3.1 Ghz with 128 Gigabytes of RAM
2002	lower bound (Eley, 2002)	lbE2002	Pentium C with 200 MHz clock and 32MB memory
2004	lower bound (Boschetti, 2004)	lbB2004	Pentium III Intel 933 MHz

Table 2.2 Lists of Papers included in the Results Comparison

No.	Box Types	Total Boxes No.	IMM1989	BR1995b	B2000	E2002-seq	E2002-sim	E2003	LZ2005	T2006	T2008	C2011	L2012	ZE2012aU	ZE2012aS	lbE2002
1	2	70	26	27	25	27	26	25	25	25	25	25	25	25	25	19
2	2	70	11	11	10	11	10	10	10	10	10	10	10	10	10	7
3	4	180	20	26	20	21	22	20	19	20	20	19	19	19	19	19
4	4	180	27	27	28	29	30	26	26	26	26	26	26	26	26	26
5	4	180	65	59	51	55	51	51	51	51	51	51	51	51	51	46
6	3	103	10	10	10	10	10	10	10	10	10	10	10	10	10	10
7	3	103	16	16	16	16	16	16	16	16	16	16	16	16	16	16
8	3	103	5	4	4	4	4	4	4	4	4	4	4	4	4	4
9	2	110	19	19	19	19	19	19	19	19	19	19	19	19	19	16
10	2	110	55	55	55	55	55	55	55	55	55	55	55	55	55	37
11	2	110	18	25	18	17	18	17	16	16	16	16	16	16	17	14
12	3	95	55	55	53	53	53	53	53	53	53	53	53	53	53	45
13	3	95	27	27	25	25	25	25	25	25	25	25	25	25	25	20
14	3	95	28	28	28	27	27	27	27	27	27	27	27	27	27	27
15	3	95	11	15	11	12	12	11	11	11	11	11	11	11	11	11
16	3	95	34	29	26	28	26	26	26	26	26	26	26	26	26	21
17	3	95	8	10	7	8	7	7	7	7	7	7	7	7	7	7
18	3	47	3	2	2	1	1	2	2	2	2	2	2	2	2	1
19	3	47	3	3	3	2	2	3	3	3	3	3	3	3	3	1
20	3	47	5	5	5	2	2	5	5	5	5	5	5	5	5	2
21	5	95	24	26	21	24	26	20	20	20	20	20	20	20	20	17
22	5	95	10	11	9	9	9	8	9	9	9	8	9	8	8	8
23	5	95	21	22	20	21	21	20	20	20	20	19	20	19	20	17
24	4	72	6	7	6	6	6	5	5	5	5	5	5	5	5	5
25	4	72	6	5	5	6	5	5	5	5	5	5	5	5	5	4
26	4	72	3	4	3	3	3	3	3	3	3	3	3	3	3	3
27	3	95	5	5	5	5	5	5	5	5	5	5	5	5	4	4
28	3	95	10	12	10	11	10	10	9	10	10	10	9	10	10	9
29	4	118	18	23	17	18	18	17	17	17	17	17	17	17	17	15
30	4	118	24	26	22	22	23	22	22	22	22	22	22	22	22	18
31	4	118	13	14	13	13	14	13	12	13	13	12	12	12	12	11
32	3	90	5	4	4	4	4	4	4	4	4	4	4	4	4	4
33	3	90	5	5	5	5	5	5	4	5	5	4	4	4	4	4
34	3	90	9	8	8	5	9	8	8	8	8	8	8	8	8	5
35	2	84	3	3	2	2	2	2	2	2	2	2	2	2	2	2
36	2	84	18	14	14	18	14	14	14	14	14	14	14	14	14	10
37	3	102	26	23	23	26	23	23	23	23	23	23	23	23	23	12
38	3	102	50	45	45	46	45	45	45	45	45	45	45	45	45	25
39	3	102	16	18	15	15	15	15	15	15	15	15	15	15	15	12
40	4	85	9	11	9	9	9	8	9	9	9	8	9	8	8	7
41	4	85	16	17	15	16	15	15	15	15	15	15	15	15	15	14
42	3	90	4	5	4	4	4	4	4	4	4	4	4	4	4	4
43	3	90	3	3	3	3	3	3	3	3	3	3	3	3	3	3
44	3	90	4	4	3	4	4	4	3	3	3	3	3	3	3	3
45	4	99	3	3	3	3	3	3	3	3	3	3	3	3	3	2
46	4	99	2	2	2	2	2	2	2	2	2	2	2	2	2	2
47	4	99	4	4	3	3	3	3	3	3	3	3	3	3	3	3
Total			763	777	705	725	716	699	694	698	698	692	694	691	693	572
Avg. Time (s)								<30	6.43	<2		45.94	6.43	246.2	116.7	

Table 2.3 Ivancic et al., 1989

Dataset	Box No.	LN1992	BMI1990	NI1994	BR1995b	GB1997	BGI1998	C1998	BGI2001	E2002	B2003	L2005	MO2005	W2008	HH2010	DD2011	R2011	L2012
		BL	D%	BL	U%	BL	U%	BL	U%	BL	U%	BL	U%	BL	U%	U%	BL	U%
LN01	100	0	78.12	0	62.5	0	62.5	0	62.5	0	62.5	62.5	28	0	62.5	62.5	7	62.5
LN02	200	32	76.77	90	54	80.7	35	90	39	89.5	28	96.6	76.02	51	89.8	53	90.8	80.4
LN03	200	0	69.46	53.4	0	53.4	0	53.4	0	53.4	0	53.4	53.43	0	53.4	0	53.4	53.4
LN04	100	0	77.57	55	0	55	0	55	0	55	0	55	54.96	0	55	0	55	55
LN05	120	1	85.79	77.2	0	77.2	0	77.2	0	77.2	0	77.2	77.19	0	77.2	0	77.2	77.2
LN06	200	45	88.55	83.1	48	88.7	77	83.1	32	91.1	49	91.2	79.51	45	92.4	44	87.9	84.8
LN07	200	21	78.17	78.7	10	81.8	18	78.7	7	83.3	0	84.7	72.14	0	84.7	0	84.7	77
LN08	130	7	67.58	59.4	0	59.4	0	59.4	0	59.4	0	59.4	59.42	0	59.4	0	59.4	59.4
LN09	200	0	84.22	61.9	0	61.9	0	61.9	0	61.9	0	61.9	61.89	0	61.9	0	61.9	61.9
LN10	250	0	70.1	67.3	0	67.3	0	67.3	0	67.3	0	67.3	67.29	0	67.3	0	67.3	67.3
LN11	100	0	65.44	62.2	0	62.2	0	62.2	0	62.2	0	62.2	62.16	0	62.2	0	62.2	62.2
LN12	120	0	79.33	78.5	0	78.5	0	78.5	0	78.5	0	78.5	71	0	78.5	0	78.5	78.5
LN13	130	15	77.03	78.1	2	84.1	20	78.1	0	85.6	4	84.3	84.14	0	85.6	0	85.6	84.9
LN14	120	0	69.09	62.8	0	62.8	0	62.8	0	62.8	0	62.8	62.81	0	62.8	0	62.8	62.8
LN15	250	0	73.56	59.5	0	59.5	0	59.5	0	59.5	0	59.5	59.46	0	59.5	0	59.5	59.5
Average			74.19	68.64	69	68.6	69.9	70.4	66.93	70.15	70.9	67	49.7	70.3	70.2	71	70.93	70.6
Avg. Time (s)																		
Stopping Time (s)						500	500	500			600						51	39.88

Table 2.4 Loh and Nee, 1992

Dataset	Box Types	BGI1998N	BGI2001	E2002	GB2002	MO2005	B2006	FB2010S	L2011	R2011	CR2012S	LIU2012	Z2012S	ZE2012aS	ZL2012S	A2014	AR2014S	L2014	MN2014	R2016S
		U%	U%	U%	U%	rt(s)	U%	U%	U%	rt(s)	U%	U%	U%	U%	U%	U%	U%	U%	U%	U%
BR1	3	92.63	87.81	n/a	88.10	8	89.07	90.57	88.14	114	93.90	94.34	94.43	93.57	94.40	81.40	94.50	90.57	93.24	93.86
BR2	5	92.70	89.40		89.56	12	90.43	90.84	89.52	145	94.54	94.88	94.87	93.87	94.85	85.70	95.03	91.46	94.10	94.55
BR3	8	92.31	90.48		90.77	25	90.86	91.43	90.53	213	94.35	95.05	95.06	94.14	95.10	87.30	95.17	92.39	93.53	94.75
BR4	10	91.62	90.63		91.03	28	90.42	92.21	94.74	308	94.08	94.75	94.89	93.86	94.81	86.90	94.97	92.33	92.74	94.63
BR5	12	90.86	90.73		91.23	40	89.57	91.25	90.75	413	94.17	94.58	94.68	93.51	94.52	86.60	94.80	92.42	92.19	94.38
BR6	15	90.04	90.72		91.28	59	89.71	91.04	90.74	500	93.48	94.39	94.53	93.39	94.33	86.30	94.65	92.55	91.56	94.24
BR7	20	88.63	90.65		91.04	64	88.05	90.81	93.20	663	92.82	93.74	93.96	92.68	93.59	85.70	94.09	92.11	90.92	93.82
BR8	30	87.11	89.73		90.26	71	86.13	92.26	88.89			92.65	93.27		92.65	85.50	93.15	91.93	89.28	93.16
BR9	40	85.76	89.06		89.50	85	85.08	91.48	88.51			91.90	92.60		92.11	84.80	92.53	91.61	89.30	92.62
BR10	50	84.73	88.40		88.73	89	84.21	90.86	87.76			91.28	92.05		91.60	84.00	92.04	91.39	88.86	92.09
BR11	60	83.55	87.53		87.87	94	83.98	90.11	87.06			90.39	91.46		90.64	82.80	91.40	91.13		91.56
BR12	70	82.79	86.94		87.18	98	83.64	89.51	86.97			89.81	90.91		90.35	81.30	90.92	90.96		91.28
BR13	80	82.29	86.25		86.70	110	83.54	88.98	86.90			89.27	90.43		89.69	80.20	90.51	90.59		90.93
BR14	90	81.33	85.55		85.81	121	83.25	88.26	86.40			88.57	89.80		89.07	78.90	89.93	90.25		90.38
BR15	100	80.85	85.23		85.48	128	83.21	87.57	86.23			87.96	89.24		88.36	78.00	89.33	89.79		90.08
Average 1-7		91.26	90.06	88.75	90.43		89.73	91.16	90.08	93.91		88.17	94.63	93.57	94.51	85.70	94.74	91.95	92.61	94.32
Average 8-15		83.55	87.34		87.69		84.13	89.88	87.34			90.23	91.22		90.56	83.70	91.22	90.96		91.51
Average		87.15	88.61		89	33.71	86.74	91.90	88.62	337		92.24	92.81		92.40	9.8	92.87	91.42		92.82
Avg. Time 1-7 (s)						68.8							500		150.55	64.7	150	67.60		146
Avg. Time 8-15 (s)								320												
Stopping Time (s)					250			60+180												

Table 2.5 Bischoff and Ratcliff (1995) and Davies and Bischoff (1999) with Stability Consideration

Dataset	Box Types	BR1995C U%	GB1997N U%	B2003 rt(s) U%	B2003iter rt(s) U%	L2003 U%	M2004 U%	L2005 U%	P2008 rt(s) U%	FB2010U U%	HH2010 U%	P2010 U%	DD2011 rt(s) U%	HH2011 U%	L2012 U%	GR2012U U%	Z2012U U%	ZE2012aU U%	ZE2012b U%	ZL2012U U%	AR2014U U%	R2016U U%
BR1	3	83.37	86.77	3	93.23	36	93.52	88.70	1.27	93.27	95.05	92.32	94.93	92.92	91.60	95.28	94.92	94.16	95.54	95.59	95.69	95.10
BR2	5	83.57	88.12	10	93.27	48	93.77	88.17	2.32	93.38	95.43	92.71	95.19	93.93	91.99	95.90	95.48	94.50	95.98	96.13	96.24	95.76
BR3	8	83.59	88.87	31	92.86	97	93.58	87.52	4.62	93.59	95.47	92.71	94.99	93.71	92.30	96.13	95.69	94.71	96.08	96.30	96.49	95.85
BR4	10	84.16	88.68	48	92.40	138	93.05	87.58	6.52	93.16	95.18	92.71	94.71	93.68	92.36	96.01	95.53	94.55	95.94	96.15	96.31	95.74
BR5	12	83.89	88.78	65	91.61	179	92.34	87.30	8.58	92.89	95.00	92.85	94.71	93.73	91.90	95.84	95.44	94.20	95.74	95.98	96.18	95.65
BR6	15	82.92	88.53	46	90.86	150	91.72	86.86	12.23	92.62	94.79	92.88	94.04	93.63	91.50	95.72	95.38	94.15	95.61	95.81	96.05	95.61
BR7	20	82.14	88.36	60	89.65	198	90.55	87.15	19.25	91.86	94.24	92.66	93.53	93.14	91.01	95.29	95.00	93.74	95.14	95.36	95.77	95.32
BR8	30	80.10	87.52						38.2	91.02	93.70	91.99	92.78	92.92	94.76	94.76	94.66		94.63	94.80	95.33	95.07
BR9	40	78.03	86.46						63.1	90.46	93.44	91.76	92.19	92.49	94.34	94.34	94.30		94.29	94.53	95.07	94.77
BR10	50	76.53	85.53						97.08	89.87	93.09	91.41	91.92	92.24	93.86	93.86	94.11		94.05	94.35	94.97	94.47
BR11	60	75.08	84.82						136.5	89.36	92.81	91.10	91.46	91.91	93.60	93.60	93.87		93.78	94.14	94.80	94.14
BR12	70	74.37	84.25						183.21	89.03	92.73	90.75	91.20	91.83	93.22	93.22	93.67		93.67	94.10	94.64	93.93
BR13	80	73.56	83.67						239.8	88.56	92.46	90.33	91.11	91.56	92.99	92.99	93.45		93.54	93.86	94.59	93.37
BR14	90	73.37	82.99						307.62	88.46	92.40	90.04	90.64	91.30	92.68	92.68	93.14		93.36	93.83	94.49	93.22
BR15	100	73.38	82.47						394.66	88.36	92.40	89.51	90.38	91.02	92.46	92.46	93.14		93.32	93.78	94.37	92.65
Average 1-7		83.38	88.30		92.00		92.70	87.60	92.94	95.00	92.83	94.53	94.53	93.53	91.81	95.74	95.35	94.29	95.72	95.90	96.11	95.58
Average 8-15		75.55	84.71						89.39	92.88	90.86	91.46	91.46	91.91		93.49	93.62		93.83	94.17	94.78	93.95
Avg. Time 1-7 (s)		79.20	86.39	38					91.05	93.90	91.78	92.89	92.89	92.67	706.59	94.54	94.53		94.71	94.98	95.40	94.71
Avg. Time 8-15 (s)									7.83	319	4661	28	10.44			147	500		500	500	500	274
Stopping Time (s)			500						5000 iter	604+180			633.37									

Table 2.6 Bischoff and Ratcliff (1995) and Davies and Bischoff (1999) without Stability Consideration

Class	Bin Dimensions	Item No.	MPV2000	MPV2000-BS	MPV2000-SPack	L2002-HA	L2002-TS	F2003	C2008	C2009	H2013-EHGH2	H2013-Ampv	lbB2004
1	100*100	50	13.6	13.5	15.3	13.9	13.4	13.4	13.7	13.4	13.8	13.6	12.9
		100	27.3	29.5	27.4	27.6	26.6	26.7	27.2	26.7	27.6	27.4	25.6
		150	38.2	38	40.4	38.1	36.7	37	37.7	37	39.8	39.4	35.8
		200	52.3	52.3	55.6	52.7	51.2	51.2	51.9	51.1	50.6	49.9	49.7
Class Total			131.4	133.3	138.7	132.3	127.9	128.3	130.5	128.2	131.8	130.3	124
4	100*100	50	29.4	29.4	29.8	29.4	29.4	29.4	29.4	29.4	29.4	29.4	29
		100	59.1	59	60	59	59	59	59	58.9	59.5	59.6	58.5
		150	87.2	87.3	87.9	86.9	86.8	86.8	86.8	86.8	90.4	90.6	86.4
		200	119.5	119.3	120.3	119	118.8	119	118.8	118.8	119	119.7	118.3
Class Total			295.2	295	298	294.3	294	294.2	294	293.9	298.3	299.3	292.2
5	100*100	50	9.2	9.1	10.2	8.5	8.4	8.3	8.4	8.3	7.9	8.4	7.6
		100	17.5	17	17.6	15.1	15	15.1	15.1	15.2	14.6	15.5	14
		150	24	23.7	24	21.4	20.4	20.2	21	20.1	21.5	23.8	18.8
		200	31.8	31.7	31.7	28.6	27.6	27.2	28.1	27.4	29.6	31.7	26
Class Total			82.5	81.5	83.5	73.6	71.4	70.8	72.6	71	73.6	79.4	66.4
6	10*10	50	9.8	11	11.2	10.5	9.9	9.8	10.1	9.8	11.8	11.3	9.4
		100	19.4	22.3	24.5	20	19.1	19.1	19.6	19.1	19.2	19.2	18.4
		150	29.6	32.4	35	30.6	29.4	29.4	29.9	29.2	29.8	28.4	28.5
		200	38.2	40.8	42.3	39.1	37.7	37.7	38.5	37.7	38.7	38.3	36.7
Class Total			97	106.5	113	100.2	96.1	96	98.1	95.8	99.5	97.2	93
7	40*40	50	8.2	8.2	9.3	8	7.5	7.4	7.5	7.4	7.4	7.9	6.8
		100	15.3	13.9	15.3	13.3	12.5	12.3	13.2	12.3	13.5	15.2	11.5
		150	19.7	18.1	20.1	17.2	16.1	15.8	17	15.8	18.2	21.2	14.4
		200	28.1	28	28.7	25.2	23.9	23.5	25.1	23.5	24.1	27	22.7
Class Total			71.3	68.2	73.4	63.7	60	59	62.8	59	63.2	71.3	55.4
8	100*100	50	10.1	9.9	11.3	9.9	9.3	9.2	9.4	9.2	9.4	10.4	8.7
		100	20.2	20.2	21.7	19.9	18.9	18.9	19.5	18.8	18.9	19.5	18.4
		150	27.3	26.8	28.3	25.7	24.1	23.9	25.2	23.9	26	27.7	22.5
		200	34.9	34	35	31.6	30.3	29.9	31.3	30	35.8	37.5	28.2
Class Total			92.5	90.9	96.3	87.1	82.6	81.9	85.4	81.9	90.1	95.1	77.8
Total			769.9	775.4	802.9	751.2	732	730.2	743.4	729.8	756.5	772.6	708.8
Stopping Time (s)			1000			1000	1000						

Table 2.7 Martello et al., 2000

2.9 Conclusion

In this chapter we have focused our review on the design and implementation of solution methodologies for solving 3D container loading problem with an experimental comparison on the performances of various algorithms on benchmark data sets. We have reviewed 113 papers that cover all variants of 3D container loading and address a wide spectrum of combinatorial optimisation methodologies, which we review according to placement heuristics, improvement heuristics and exact methods. We have attempted to provide an insightful review that describes the main ideas clearly. These sections do not differentiate between problem types, so we provide an overview of these papers by problem variant. Our examination of the literature has highlighted three possible fruitful avenues for research. First we found that there are significantly fewer papers examining the multiple container packing problem when compared to the single container loading problem. Further, of the papers looking at multiple containers, very few consider heterogeneous container problem types. Another observation is that often papers that tackle the multi container problem do so by simply extending their models for the single container loading problem. A second research issue surrounds the consideration of real world constraints. Only a small proportion of papers comprehensively address this, while most other papers focus on a few specific constraints. Even within those papers discussing real world constraints, not all critical constraints are taken into consideration. A consistent set of real world constraints would benefit the research community and make adoption by practitioners more likely. This links with the final research issue regarding the quantity and quality of benchmark datasets. As Bortfeldt and Wäscher (2013) point out, the current decade-old data sets might not be challenging anymore. Realistic and challenging data sets with clear real world constraints would help in moving this research area forward.

Chapter 3

Methodology Review

3.1 Introduction

During the review of 3D container loading algorithms in the last chapter, we noticed some improvement heuristics are popularly implemented. In this chapter, we review these commonly applied improvement heuristics in the area of 3D container loading problem. We will also review beam search, a tree search algorithm often applied in scheduling problems and recently in irregular strip packing problem. We intend to implement beam search on our specific 3D container packing problem for the first time.

The chapter is structured as follows. Firstly, the general concept of heuristics and metaheuristics are briefly discussed. We then give description on improvement heuristics Descent Method, Iterated Local Search, Genetic Algorithm, Simulated Annealing, Tabu Search, and other less commonly applied ones such as Greedy Randomised Adaptive Search Procedures, Variable Neighborhood Search and Guided Local Search. We explain beam search in detail as the last search algorithm. At last, We will conclude our decision on the improvement heuristics we choose to implement for our specific multiple containers problem.

3.2 Heuristics and Metaheuristics

Reeves and Beasley (1995) defines heuristic as "a technique which seeks good (i.e. near optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality or even in many cases to state how close to optimality a particular

feasible solution is". Many heuristics consist of a combination of the four basic strategies classified for heuristic procedures (Foulds, 1983).

construction strategy: starts with a partial solution. Potential valuable new elements are added successively to the final solution. This is a useful strategy when generating feasible starting solutions is difficult.

improvement strategy: starts with a sub-optimal solution and progressively improves through modifications. This strategy is suitable when generating starting solutions is easy.

component analysis strategy: firstly divides the problem into component parts. These parts are then optimised separately and recompiled in the end.

learning strategy: adopts a tree-search approach similar to branch & bound. The outcomes of earlier decisions guide the choice of future branches.

When the heuristics search becomes trapped in a local optimum, meta-heuristics are introduced to escape such local optimal points. Osman and Kelly (1996) defines meta-heuristic as "an iterative generation process which guides a sub-ordinate heuristic by combining intelligently different concepts for exploring and exploiting the search spaces using learning strategies to structure information in order to find efficiently near optimal solutions".

3.3 Descent Method

Descent method only allows improving moves. Any deteriorating or neutral move is rejected. Finding no more improving moves indicates the termination of the procedure. Three general steps are described below. The method is mediocre in a way that it usually only reaches a local optimum rather than further seeking a global optimum. Additional acceptance rule which allows non-improving moves can be introduced to allow further exploration outside of current local optimum. Because of the limit of descent method, various meta-heuristic methods are developed.

Step 1: Choose an initial solution s_0 , and set $s = s_0$.

Step 2: Find a solution s^* in a set of neighboring solutions $N(s)$ for which $F(s^*) \leq F(s')$ for any $s' \in N(s)$.

Step 3: Terminate if $F(s^*) \geq F(s)$. Otherwise set $s = s^*$ then go to **Step 2**.

3.4 Iterated Local Search (ILS)

Local search is a metaheuristic method for solving optimization problems that are computationally hard. Local search can be applied on problems for which a solution is to be found among a number of candidate solutions. Local search algorithms move from solution to solution among the search space of candidate solutions till a solution considered optimal is found or a stopping time is reached. Iterated local search (ILS) generates a sequence of solutions and each such solution is generated by the embedded heuristic within the ILS. The parameter-less approach was first applied by Baxter (1981) to a location problem and Baum (1986) to the traveling salesman problem (TSP). Since then ILS has been widely applied to TSP, scheduling problems, and cutting and packing problems along with many industrial problems successfully. The notation of ILS can be presented as follow. S represents the set of candidate solutions. s represents solution in S . F represents the cost function. $F(s)$ represents the cost function value of solution s . s^* represents locally optimal solution. The mapping from S towards S^* is achieved by local search. When having an initial solution s_0 , local search randomly restarts each time and generate multiple s^* independently. A smaller space S^* from a large space S is the result of local search. Within S^* perturbation is applied to a given s^* resulting a random move towards other neighbourhood from s^* to s' , and then local search is applied to generate a s_{new}^* as shown in figure 3.1. The new s^* is decided by acceptance test. The procedural view of iterated local search is shown below.

Step 1: Generate initial solution S_0 and compute cost function $F(s_0)$.

Step 2: Apply local search to reach local optimal from s_0 to s^* for which cost function $F(s^*) \leq F(s_0)$.

Step 3: Repeat following steps.

Perform perturbation to randomly move towards other neighbourhood from s^* to s' .

Apply local search to reach local optimal from s' to s_{new}^* for which cost function $F(s_{new}^*) \leq F(s')$.

Compare $F(s^*)$ and $F(s_{new}^*)$ to decide the new optimal s^* (in some cases a ‘wait and see’ strategy is applied) according to acceptance criterion.

Step 4: Stop until termination condition is met (usually in term of exceeding the preset computation time or completing the preset number of iterations).

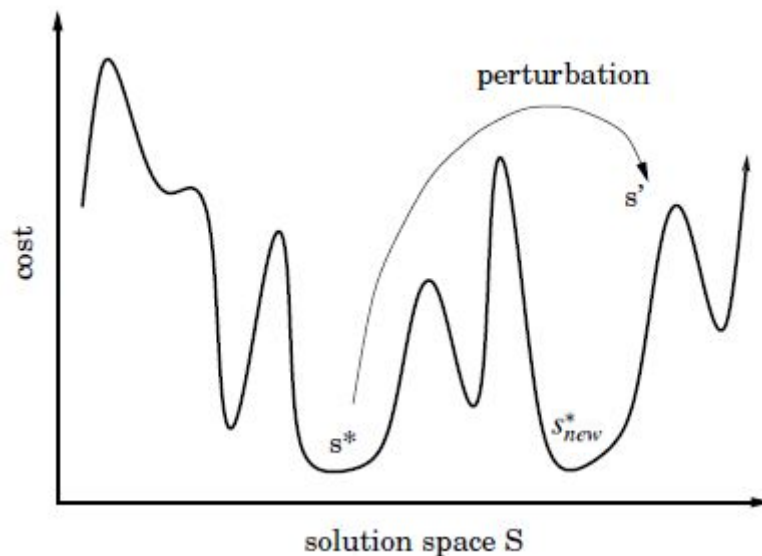


Fig. 3.1 Pictorial representation of iterated local search

The basic framework of iterated local search often leads to considerably good performance and only requires a few lines of additional code. It has a nature of modular approach which steps can be added one at a time to increase complexity. The local optimisation of ILS can be achieved through optimising single modules with other modules remaining unchanged. Some guidelines of ILS are described below.

Though dependence on initial starting point s_0 is very low for long runs, the initial solution is preferable to be greedy rather than random. The strength of perturbation is important because a strong perturbation is close to random restart and a weak perturbation may not be strong enough to jump out of local optimal in which case applying local search may undo the perturbation. The design of local search should be as effective and as fast as possible. The choice on perturbation depends strongly on the design of local search. If local search has known short-comings, a good perturbation should compensate for those short-comings. Execution on local search with small perturbations is much faster as fewer elements are changed.

Acceptance criterion strongly influences the effectiveness of reaching to the optimal as an aggressive acceptance criterion may or may not help speed up the search process. The choice of acceptance criterion depends strongly on perturbation and local search. The interactions between perturbation strength and acceptance criterion is important as well. There is always a trade-off between intensification and diversification. An extreme intensification is when the new solution is only accepted if the new solution improves on the historical

solutions in term of functional cost. An extreme case for diversification is to accept the new solution no matter what. The combination of perturbation and acceptance criterion controls the relative balance of intensification and diversification.

Search history is also very useful in iterated local search. An example of using history is that the memory about the previously found local minima is there to help find better and better starting points for local search. Some known local search algorithms are iterative improvement, threshold accepting, simulated annealing and tabu search. Overall, since the performance of a local search algorithm highly depends on the construction of the neighbourhood, a method, if one exists, designed based on the specific structure and nature of a particular problem is usually preferable.

3.5 Genetic Algorithm

Developed by Holland (1975), a general genetic algorithm (GA) brings higher performance, can be easily implemented and prevents the occurrence of duplicate descendants. A general procedure of GA is presented below. Different from shown in the procedure, it can also alternatively adopt a crossover or a mutation operator. Two complementary probabilities, which have the sum of 1, decide the choice of the two operators. If no crossover is performed, the children are the exact copy of parents. The survival of the best solution is guaranteed during the entire re-production process through replacing weak individuals in the current population with descendants generated.

Step 1: Generate an initial population of n chromosomes.

Step 2: Evaluate the fitness $f(s)$ of each chromosome s .

Step 3: Create a new population by repeating these processes until completion.

Selection: Choose 2 parent chromosomes based on their fitness, and the fitter ones have higher chance to be chosen.

Crossover: Cross over the parents to form 2 children according to the crossover probability p_{cross} .

Mutation: Mutate children at each position in chromosome according to the mutation probability p_{mut} .

Replace: Replace children for those with lowest fitness.

Step 4: Evaluate the fitness $f(s')$ of each chromosome s' in the new population.

Step 5: Return the best solution and terminate algorithm if stopping criterion or goal is reached, otherwise go to **Step 3**.

A genetic algorithm is considered to be suitable for large and hard optimization problems such as three dimensional container packing based on several major characteristics. GA guides a highly exploitative search from a population of points, not a single point, through a parameter space based on random choice. It uses payoff or objective function information, not derivatives or other auxiliary knowledge. Probabilistic transition rules, not deterministic rules, are used.

The hybridisation approach is summarized in the equation $GAs + data\ structures = evolution\ programs$. Local search is applied during the evolutionary cycle. The solutions of hybrid GA are presented as more complex data structures such as graphs or matrices. The actual problem needs specifically-designed operators to generate the offspring.

3.6 Simulated Annealing

Firstly purposed by Metropolis et al. (1953), simulated annealing started to draw interest through the work of Kirkpatrick et al. (1983) and Černý (1985). The method is a local search where a move to an inferior solution is controlled by an acceptance probability that decreases along with the progression of the search. The simulated annealing algorithm for maximization problems is presented below. The movement to inferior solutions allows the algorithm to escape from local optima by exploring unknown regions of the search space. The need to leave a local optimum is high at the early stage of the search process. One of the advantages of SA is that it can move quickly from current solution to another feasible solution once the acceptance of a randomly selected neighbor has been confirmed. However, it also means that promising neighbors may be omitted too easily.

Step 1: Initialise solution $s = s_0$ and temperature $t = t_0$.

Step 2: Local search, and derive a neighbour s' in neighbourhood $N(s)$ and energy $\Delta E = E(s') - E(s)$.

Step 3: If $\Delta E < 0$, then set $s = s'$.

Step 4: Else generate random number $x \in [0, 1]$, and if $x < e^{-\frac{\Delta E}{t}}$ then set $s = s'$.

Step 5: Reduce temperature t by cooling function $t = C(t)$, and go to **Step 2**.

Step 6: Terminate algorithm when reaching stopping criterion or goal.

3.7 Tabu Search

Originally developed by Fred Glover (1986, 1989, 1990), a general tabu search algorithm is shown below. Tabu list contains a list of solution points or move attributes that are not allowed, while aspiration criteria allow exceptional moves which lead to promising solutions from the tabu list. The structure of the tabu list prevents cycling through solutions as the move from a new solution to the last solution is recorded and banned for a certain rounds. However, a move found in the tabu list is still available for run if it leads to a new global best solution. TS has the ability to intensively examine regions of the search space by visiting all neighbors of a given solution, though computing time is at the risk to be wasted if no global optimal (or near optimal) solution exists in the current region. Based on the specific problem and algorithm, the tabu list may contain different number, size, contents, and management policies. A method named diversification may be performed to enhance the tabu search by guiding the search towards zones of the unexplored solution space.

Step 1: Generate initial solution.

Step 2: Create a candidate list of moves.

Step 3: Choose the best admissible candidate based on the tabu list and aspiration criteria.

Step 4: Record it as the new best solution if it improves on the previous best.

Step 5: Stop if stopping criterion is reached. Otherwise go to **Step 2**.

3.8 Greedy Randomised Adaptive Search Procedures - GRASP

GRASP algorithm, an iterative procedure, first developed by Feo and Resende (1989) then improved by Resende and Ribeiro (2003), consists of a constructive phase and an improvement phase. First a solution is generated within the constructive phase. The solution is built by adding element to a partial solution at each step. A greedy function is adapted to choose the elements to be added. Different solutions can be obtained through the adoption of a randomization strategy at each iteration. A simple local search is usually carried out in the improvement phase.

3.9 Variable Neighborhood Search - VNS

Mladenović and Hansen (1997) develops Variable Neighborhood Search (VNS), a meta-heuristic procedure, to solve hard combinatorial problems such as graph problems, location, routing, sequencing and scheduling problems. VNS algorithm changes neighborhoods systematically by exploring increasingly distant neighbourhoods of the current solution. Among a set of neighbourhoods, VNS starts at the first neighbourhood. Any local search method could be used to explore within the current neighbourhood. The current solution is updated if a better solution is found. If no better solution can be found in the current neighbourhood, the search stops within the current neighbourhood and a shaking phase is performed to diversify the search to the next neighbourhood. Such process terminates when a stopping criterion is met. The philosophy of VNS is to consider more than one neighbourhood to avoid being held by local minimum.

3.10 Guided Local Search - GLS

Among all the local search methods, Guided local search (GLS), a powerful metaheuristic for solving hard optimization problems (Voudouris, 1997; Voudouris and Tsang, 1999), may be classified as a tabu search heuristic since the way it uses memory to control the search is similar to that of tabu search. Search is guided to promising regions of the solution space based on memory as some features of previously visited solutions are labeled as bad by augmenting the cost function with a penalty term. The local search is regarded as a process of iteratively improving towards local optimisation. A solution is characterized by a set of features which all have more or less direct contribution to the value of solution upon their presence. The contribution is defined by the cost assigned to the feature. Penalty may be given to the feature with a high cost. The neighbourhood is divided into smaller sub-neighbourhoods. Fast local search (FLS) is further implemented to drastically speed up the local search towards a local minimum by de-activating less promising sub-neighbourhoods based on the lack of improving move found.

3.11 Beam Search

Unlike all the heuristics mentioned above which always generate an initial solution, beam search adds a new element to the partial solution through each step and a final solution is

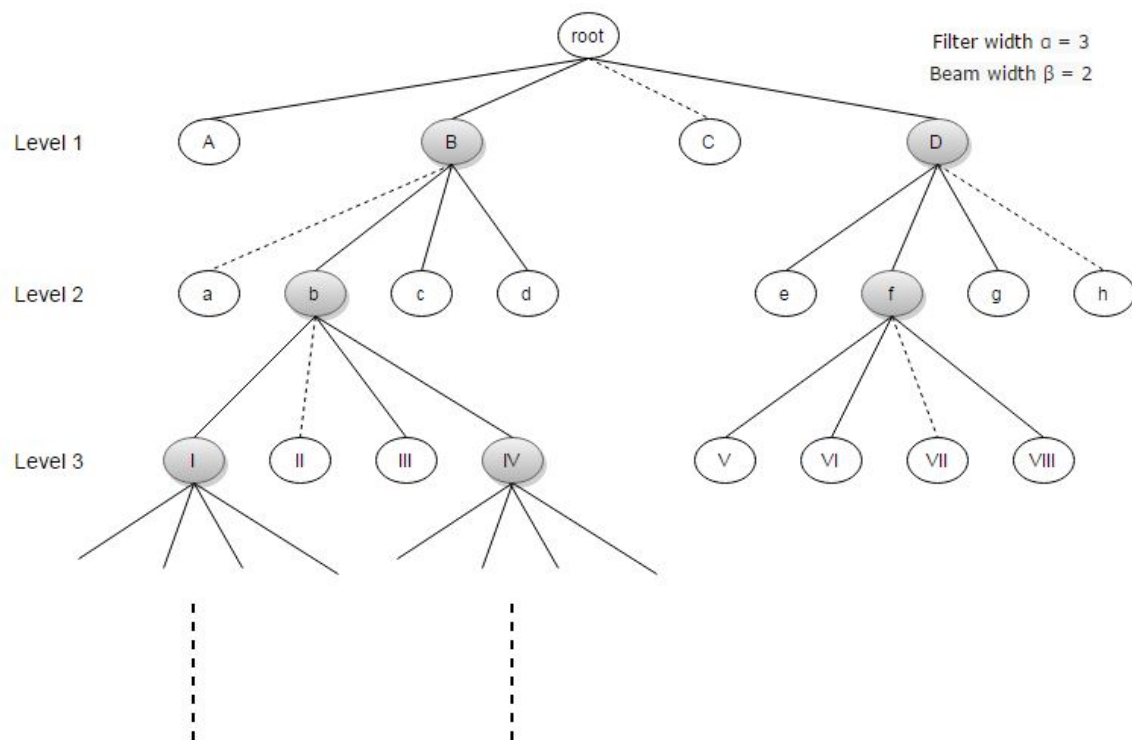


Fig. 3.2 Illustration of a beam search

generated at the end of the whole procedure. Similar to branch-and-bound, beam search uses a tree search structure of nodes and branches. Common application of beam search can be found in scheduling problems (Sabuncuoglu and Bayiz 1999, Blum 2005, Ghirardi and Potts 2005) because of the nature of its structure lending itself to modeling problems where sequences are constructed. The search is breadth first, and an evaluation is given to aggressively prune the branches at each level. The number of branches retained is user-defined, thus the running time of the algorithm can be controlled given the polynomial nature of its size. A combination of local and global evaluation functions is used to decide the most promising nodes, and only those nodes are to be branched from in the next level. A local evaluation can be fast but may face the risk of discarding good solutions, while a global evaluation may cost much more computational time but is more accurate to predict which nodes are promising. In general, beam search does not allow backtracking. However, Ghirardi and Potts (2005) designs recovery strategies to allow beam search to recover from wrong decisions. The basic structure and its variant of beam search is described below.

Figure 3.2 illustrates the structure of beam search. Each node adds a new element to the partial solution. Root node contains the initial partial solution, and a complete final solution is formed in the last level at the bottom of the tree. All child nodes at each level are evaluated

and the approximate solution quality is predicted by a local evaluation function. During the local evaluation, only child nodes from the same parent node are competed with each other. This is because the local evaluation function only measures the cost of adding an element to the current partial solution without taking account of the elements which are not included in the partial solution yet. The filter width α is user-defined, and the best α nodes from the same parent node are selected after local evaluation. Global evaluation is then carried on these filtered α nodes. During the global evaluation, all selected filtered nodes from different parent nodes (i.e. beams) are competed with each other. The beam width β is user-defined, and the best β nodes are retained for branching in the next level. An example of such procedures is demonstrated in figure 3.2 with user-defined filter width α as 3 and beam width β as 2. At Level 1, there are four possible child nodes, i.e. A, B, C, D, derived from the root node. Local evaluation is carried on all four nodes. The best three nodes A, B and D are selected, and node C is pruned as illustrated in broken line. The three filtered nodes are then evaluated using global evaluation function, and the best two nodes B and D are selected as highlighted in the figure. Node C is pruned. Therefore, node B and D serve as parent nodes for Level 2. At Level 2, four child nodes are derived from each parent node. That is child node a, b, c and d from parent node B, and child node e, f, g and h from parent node D. Local evaluation is carried on all 8 child nodes. The best three child nodes b, c and d are selected under parent node B, and child nodes e, f and g are selected under parent node D. The six filtered nodes are then evaluated globally, and the best two nodes b and f are selected to be parent nodes for Level 3. It is possible that global evaluation selects all its beam nodes from the same parent node as shown in Level 3. Child nodes I and IV are selected as beam nodes from the same parent node b. The branches derived from the parent node f are totally pruned in this case. It is clear that user has a higher chance to find better solutions by setting the values of α and β larger in the cost of computational time.

3.12 Conclusion

In this chapter, we reviewed those improvement heuristics which are commonly applied in the area of 3D container loading problem. The reason we totally exclude exact method in our consideration is because our sponsor Gower Optimal Algorithms Ltd (GOAL) does not want to buy a commercial solver. Even we can model it, it is unlikely to solve our problem given the number of boxes we are trying to pack. Among the improvement heuristics we reviewed above, we choose Iterated Local Search (ILS) and Beam Search (BS).

We choose ILS because of its simple and few parameters characteristic. As we have lit-

the control over our sponsor's constructive heuristic, improvement heuristics with lots of parameters may not be the best fit because those heuristics require knowledge on constructive heuristics to change parameters for reaching better results. Also, under the modular approach of ILS we are able to freely add or remove steps for better results. During the implementation of Iterated Local Search (ILS), we adopt the improvement strategy we discussed in section 3.2 as it is easy to generate a feasible initial solution of decent quality with the help of our sponsor's constructive heuristics. An initial sub-optimal solution (or partial solution which is very close to a full solution) is generated for future improvement. We also adopt Descent Method within one of our ILS algorithms. Given the complexity of the problem we are trying to solve, we would try to get closer to global optimum through necessary deteriorating or neutral move. It would be interesting to see how improvement heuristic would perform when only improving moves are allowed.

Beam Search (BS) is the other improvement heuristic we implement. It is a combination of construction strategy and learning strategy we discussed in section 3.2. While generating a feasible initial solution can be easily done in our case, it may be the case that the initial solution contains poor choices needed to be corrected through progressive modifications. In this sense, Beam Search works in a different way compared to Iterated Local Search. It allows a few partial solutions to be generated in parallel and builds gradually towards the final solutions with the best new elements. We implement BS to allow our sponsor's constructive heuristics to do what they are best at. We do not correct any poor choice we think they might make. Instead, we only choose the best ones.

Chapter 4

Gower Optimal Algorithms Ltd (GOAL)

4.1 Introduction

Our project is sponsored by Gower Optimal Algorithms Ltd (GOAL), a software company providing logistics solutions. GOAL develops software products for pallet loading, product and packaging design, and container and trailer loading with the promise to reduce warehousing, transportation and packaging costs for small and medium-sized enterprises and multi-national organisations. Our specific project is on solving multiple (homogeneous and heterogeneous) containers problem. GOAL provides us their software Cargo Manager (CM) which is comprised of constructive heuristics specialising in packing a single container along with taking care of all the restrict industry constraints. Their requirement to us is that we embed CM as a standalone within our algorithms so that the solutions we come up with are feasible and by GOAL's standard. The consequence of such requirement on our algorithm design is that we build our search algorithms around embedded CM and run CM every time when we evaluate a solution's feasibility and quality.

In this chapter, we intend to draw a brief introduction on our sponsor Gower Optimal Algorithms Ltd (GOAL) and their single container packing software Cargo Manager (CM). We will place emphasis on explaining how it is 'conservative' in what it allows making CM appear uncompetitive. We will also briefly discuss any difficulties we have to overcome when dealing with a third-party software. Moreover, we will investigate CM and carry out analysis of CM algorithm's performance. Such analysis includes algorithm execution time

under different conditions and the impact of different sequencing and heuristics.

The structure of this chapter is as follows. In section 4.2, we will briefly introduce our sponsor, Gower Optimal Algorithms, along with the constraints they face in the industry with a vivid example of their ‘conservativeness’. We then describe how the CM algorithm works in general and its constructive heuristic methods along with challenges we faced when trying to embed CM into our own algorithms. In section 4.4, we discuss on what aspects we investigate CM and present our new benchmark data sets which are essential for all our future experiments. Experimental results on the investigation are presented in section 4.5. A short conclusion is drawn at the end of the chapter.

4.2 Gower Optimal Algorithms

The efficient use of containers benefits both small and medium-sized enterprises and multinational organisations economically in the area of production and distribution of goods. A high container utilisation can save considerable cost. Our sponsor Gower Optimal Algorithms provide computer supported packing methods to achieve this goal. Cargo Manager is one of their flagship products designed to pack containers more efficiently. As shown in figure 4.1, customers can input box dimensions and quantity along with other constraints they want to define for each box type. A visual solution is presented for customer use.

Common constraints reviewed in the academic literature mean more than just a few extensions to our sponsor. Unlike academic which always sets performance improvement as the first priority, industry would appreciate something never fails over a not guaranteed better performer. Many clients use Cargo Manager as a part of the quotation procedure to have a good estimation of the number of containers needed for an assignment. Therefore, any inaccurate initial measurements may create problems. Even when the actual product is smaller than that quoted, the already generated packing pattern may not be accurate enough as a guide for actual packing. Unfortunately they face a more dynamic world where the internal load space varies between operators and even within companies.

Note that extra constraints are to be considered when pallets are used in container loading. On many occasions, identical cargo are required to be grouped together so that it is easier to be palletised and unloaded. Cargo (i.e. boxes, which is the most commonly used name by researchers) are firstly arranged onto pallets before pallets are then packed (usually involving stacking onto each other) into containers. While the flat structure of pallet is able to provide cargo with stable support from the bottom, extra measures are needed for stability

Help About Exit Restart Table ContDims Contbase Manual ItemSearch Itembase Help

Itembase

TRIAL 2 PROMOTION
TRIAL 3 BULK
TRIAL 4 CATERING

Double Click item to enter it into CURRENT input screen

Use with Shift or Caps Lock to Insert as a New Cargo entry

To add items to the Itembase select it on Screens 1 or 4 (see Help or Manual Sect 10)

Search Itembase

Build up your Cargo List until it is complete and then select **Cargo List Complete**
Data may be entered using the keyboard, or by double clicking in the Itembase Database window.
Keyboard entry of a Case code/description matching a database entry will auto-complete other fields.
[The Database can be updated on Screen 4 using data entered here, or otherwise from Screen 1 - see Manual]

Case identifier: A Case code/description: TRIAL 1 CONTAINERS
(Item No. 1 of 4)

Case dimensions: Length 1100 mm Width 750 mm Height 1550 mm Cylinder? ☐ Yes

Permitted orientations -
Can be placed vertically: ☐ Yes ☐ Yes Yes
Must be placed lengthwise: ☐ Yes ☐ Yes No

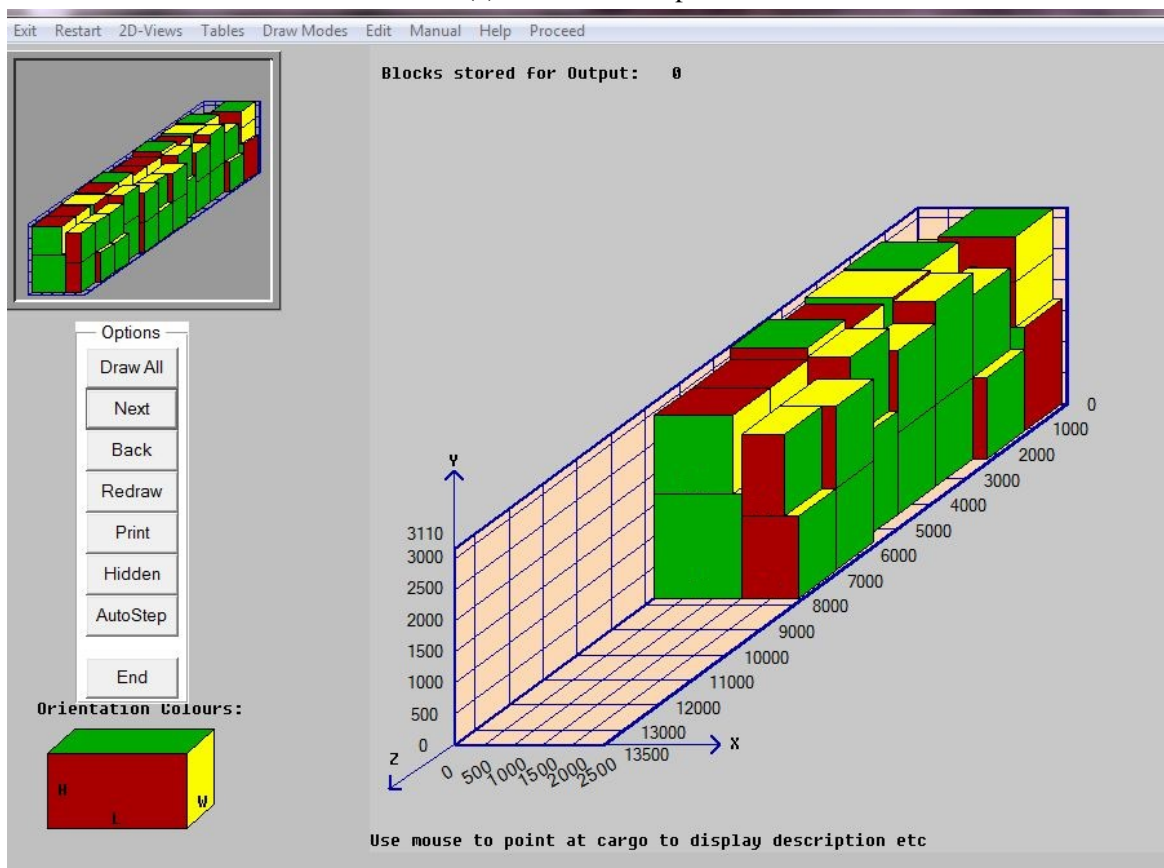
Case weight: 500 kg [Volume of each Case: 1.279 Cu.M]

Item must be placed on floor: ☒ Yes Number to be packed: 3
No other case types on top: ☐ Yes Packing priority (normally 1-99): 1
Max. number in stack: 1 [Total Volume of this Case type: 3.836 Cu.M]

Total Number of Cases in the Consignment: 143 Total Volume of Consignment: 23.632 Cu.M
Ratio: Total Cargo Volume / Container Vol.: 0.65 Total Volume of the Container: 36.110 Cu.M

Edit/Display Items Edit Cargo List Pack
++ Next Previous -- Search Add more Items Delete this Item Cargo List Complete

(a) Information Input



(b) Solution Output

Fig. 4.1 Cargo Manager

due to the absence of support from sides (Bischoff and Ratcliff, 1995). Moreover, the current Europe pallet sizes of $1200mm \times 1000mm$ (UK) and $1200mm \times 800mm$ (Euro) do not fit very well with the footprint of most container sizes.

Facing a more robust container loading world than that of academic, a few common extended measures taken by our sponsor are sampled below.

Stability Overhang over the front of the completed load is not desirable in most cases. Some clients have requested building a wall by interlocking blocks like bricks. However, this leaves the question of what to do at the edges.

Load bearing In the case of light squashable products the case itself provides the load bearing strength. It will not support a smaller heavier product placed at its centre.

Gravity centre Calculations may become complex for road transport as the load on each axle is included into consideration. The 'Load Levelling' function in CM is used to spread the completely loaded cargo out.

Access/loading This is an important issue for multi-drop situations. Due to the load stability constraints, the arrangement of some loads can not be placed at their ideal positions. Therefore, trailers with 'curtain sides', which ensures items be accessed along the whole length of the container, are used so it may be possible to place an early drop in the middle and to maintain the gravity centre after it's removed.

Product quality/compatibility The storage requirements of some loads may not be homogenous, and food is an example requiring different temperature during transport. In some cases, products in the same loads can not be packed together, such as food and cleaning products. In other cases, some products must not be in the same container, such as kiwi fruit and apple. Or additional neutral products are used as the buffer to avoid incompatible products being too close. Other than these, empty spaces need to be left out around some products such as frozen goods to ensure product quality.

Consignment Items related to each other are packed in the same consignment. Examples are parts which will be assembled together, and related items of clothing which will be sold as a single outfit. In the case that the whole consignment cannot be packed, the correct proportions of related items should be packed instead. For cases that a large consignment is to be loaded into several containers in an on-line situation, buffer space to store products prior to packing or the number of open containers at a time may be limited. As a result, the packing order usually resembles the production order closely.

Cost of Loading and Unloading There will always be a trade-off between efficiency of packing and the cost of loading and unloading especially in the case of a mixed load with many small items. Larger cases or pallets are used first for these small items. In the case that boxes goods are first packed into a small set of boxes before those boxes being packed into container, decision is made on which box is used for which product.

As a result, our sponsor tends to adopt a fairly conservative approach with respect to permitted box arrangements when compared to the methods in the literature. We hereby describe an example to explain the conservativeness implied by our sponsor. Given a situation where boxes within the same consignment must be packed in the same container. The instance is taken from the Vehicle Routing and Loading Problem (VRLP) data set of Gendreau et al. (2016). The original data set does not include any unit measurement on either container or box dimension. We therefore duplicate the original container size and box sizes as shown below.

Container size: length 60, width 25, height 30;

Consignment 1:

Box 1: length 32, width 15, height 11;

Box 2: length 33, width 12, height 13;

Box 3: length 14, width 10, height 9;

Consignment 2:

Box 4: length 21, width 12, height 15;

Box 5: length 34, width 12, height 13;

Methods in the literature have no problem packing each consignment into the same container. Using our sponsor's algorithm, all the boxes in consignment 1 can only be packed in the same container if consignment 2 is packed first as shown in figure 4.2a. However, if consignment 1 is packed alone or packed first it only packs box 1 and box 3 as shown in figure 4.2b. This is because placing box 2 either under or above box 1 results in an overhang, and certain situations overhang is not allowed in order to meet practical needs. The first situation is when the overhang might block off a significant container space for later packing. Especially in the case of filling spaces above the floor when the floor has not yet been filled. The logic behind this application is complicated, as told by our sponsor. The other situation is that when packing near to the container door our sponsor ensures a reason-

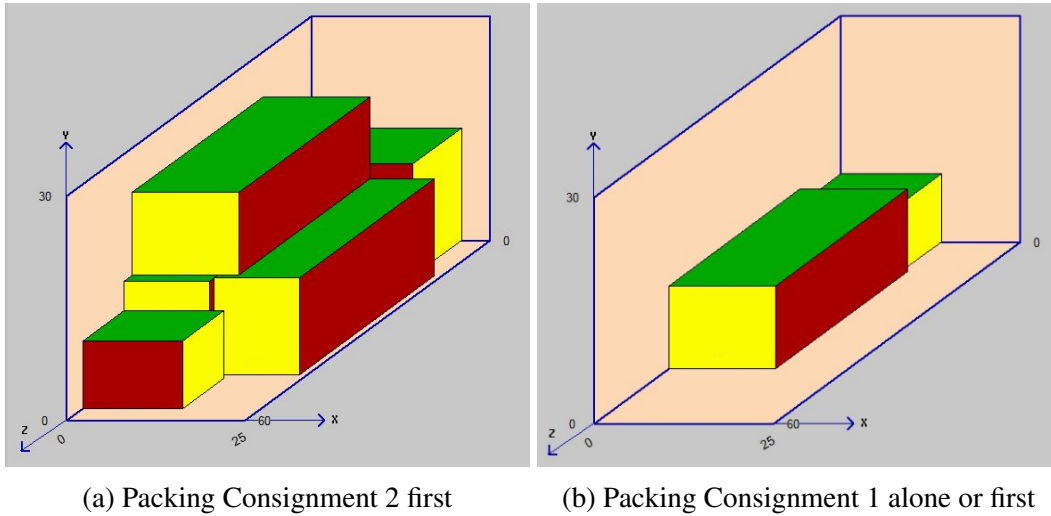


Fig. 4.2 An Example of Cargo Manager's Conservativeness

ably neat and stable packing face. The above infeasible solution is caused by this second situation. In practice container loaders do not like or use overhang in which only part of an item is supported.

One reason is that overhanging boxes are likely to cause crushing of the boxes especially towards the front of the container. According to GOAL, geometrical arrangement applying overhang strategy permitted in academic papers may not always meet practical needs. Furthermore, directly placing a small box in the middle top of a much larger box is undesirable due to the lack of support from the sides of the larger box. Our sponsor tackles this problem with an option to strictly generate piles of only identical size.

4.3 Cargo Manager

Here we will help draw an understanding on Cargo Manager (CM). Firstly, we will describe how CM works in general. Then we will explain CM's various constructive heuristics in more detail. In section 4.3.3, we will brief discuss the challenges including technical ones we face when embedding CM into our own algorithms.

4.3.1 How Cargo Manager Works

Cargo Manager (CM) is a stand alone executable which packs a single container. Packing starts at the back of the container and works through the items in the sorted order attempting

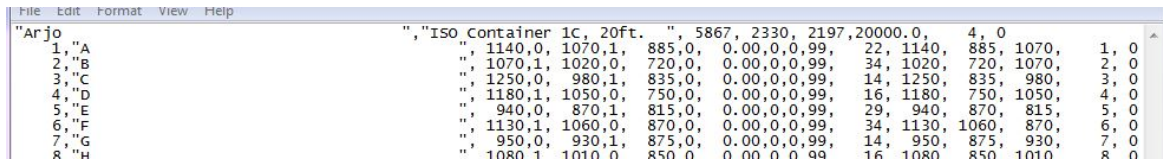
to place them using a series of placement rules. If an item is not suitable then the next item is tried, with the 'unsuitable' item being considered again for the next placement. As the data is sorted into priority order, higher priority items will tend to be packed first. If it is a requirement that no item of the next priority is packed unless all items of the current priority have been packed first, then we are able to set the packing as priorities strictly applied. For a straight-forward problem with no multi-drop requirement, CM executes a main stage and an additional stage.

The main stage attempts to place the boxes using different constructive heuristic methods, from the very basic heuristic to the more time-consuming ones. Boxes are placed in blocks if possible, where a single block consists of boxes of same type with the same orientation. All the methods are based on the idea of filling spaces. At the start, there is a single space consisting of the whole container. As each item or block of items is placed, the space in which it is placed is no longer available and a set of new cuboid shaped spaces are created. Some of the new spaces may be merged with previous spaces. All heuristic methods select the boxes from the current list in order. Each feasible depth-wise dimension for the box is tried, and the largest possible block is placed in one or more of the available spaces. The heuristic methods differ in the way of selecting the spaces and the boxes and restrictions put on the depth of a block. In the next section, we will discuss in more detail on the five fundamental constructive heuristics GOAL builds into Cargo Manager. In this main stage, as many items as possible are packed in a series of 'walls' designed to make the maximum possible use of the rectangular cross-section defined by the height and width of the container.

In the additional stage, the very basic heuristic from the main stage is used to pack as many of the remaining items as possible into the available spaces. The additional stage is not suitable for multi-drop problems or problems where load leveling is required. In this stage heavy or fragile items are also excluded from the wall-building routine.

Solutions produced by different constructive heuristic methods are evaluated based on volume packed and then on length used in the case of equal volume. Let $V(X)$ be the volume of boxes packed in solution X and $L(X)$ be the length used by solution X . Solution A is judged to be better than solution B if $V(A) > V(B)$ or $V(A) = V(B)$ and $L(A) < L(B)$. The best solution is chosen as the final solution of the run. User is able to decide if all or only a few heuristics are to be involved. In the case of only one selected heuristic method, its solution automatically becomes the final solution.

The input file read by CM is in the form of text document (.txt) as shown in figure 4.3. The first row contains container dimensions. From the second row onward, each row contains



File	Edit	Format	View	Help
"Arjo				
1, "A				
2, "B				
3, "C				
4, "D				
5, "E				
6, "F				
7, "G				
8, "H				

Fig. 4.3 Input File for Cargo Manager

details of a single box type including dimensions, box number, and other constraints such as weight if provided. Each row is considered as a unique box type even two or more box types may contain exact same details, i.e. same dimensions, quantity and constraint. We define all the boxes in a row as a batch. This concept will be frequently referred to during our modeling in chapter 5 and 6. A unique index is given to each batch to distinguish themselves from each other. During packing, CM will give higher priority to the batch listed ahead. The position of a batch indicates its priority. In our example, batch A has the highest priority and batch H has the lowest priority. The output files generated by CM are also in the form of text document (.txt). One piece of information very useful to us is the remaining box number of each batch. After each run of CM, we replace the original box number of each batch with this information before another run of CM if there are still boxes left. The impact of different box types and box numbers on the CM execution time is presented in section 4.5.1.

4.3.2 Constructive Heuristics

As we discussed earlier, GOAL's Cargo Manager (CM) executes heuristics set by users, and selects the best solutions by comparing each of them. There are 16 separate heuristics overall, while 5 of them serve as basic heuristics. Brief descriptions of these 5 heuristics are provided below.

Herusitic 0 (H0) is the basic heuristic. It packs from the back and uses as much of the depth, height and width as possible.

Herusitic 2 (H2) considers the possibility that swapping the height/width dimensions of the boxes may allow for another block of the same type of box to placed to the side and selects the best option.

Herusitic 3 (H3) is similar to H2 but instead of working with blocks of maximum depth it uses blocks of depth of 1 box. This means that orientation choices are made more often possibly resulting in better packings.

Herusitic 4 (H4) is similar to H3 but biases the choice of spaces to try and fill across the width of the container before moving forward.

Herusitic 5 (H5) is similar to H2 but biases the choices towards spaces on the floor and was originally conceived for the loading of mixed pallets where the actual physical loading operation is from the base up. This is the only layer building heuristic compared to all the other wall building ones.

While Heuristics 0, 2, 3, 4 and 5 do not allow boxes to project over the boxes that support them, Heuristics 10, 12 and 13 apply the same strategies as H0, H2 and H3 but allowing overhang up to a percentage limit set by the user. The other eight heuristics, namely S0, S2, S3, S4, S5, S10, S12 and S13, apply H0, H2, H3, H4, H5, H10, H12 and H13 with the length and width dimensions of the container swapped.

4.3.3 Challenges in working with Cargo Manager

We think it is important to mention the challenges we face when working with Cargo Manager. We face not only challenges in working in its context but also technical challenges. Unlike most literatures which structure their own placement heuristic, we have to embed Cargo Manager within our own search heuristics. Therefore, extra coding is done just to transfer input and output between the standalone and our algorithms. We also had difficulties to have Cargo Manager to run on the university computing cluster. As Cargo Manager can only be run on the Windows environment, the university computing cluster runs on a linux system. This was a major woe for us that we had to wait for week for the results which could be obtained in minutes by running in parallel on the university computing cluster. The other difficulty we faced was that it took quite a while to have a final version of Cargo Manager. As the research went, our sponsor made several modifications along the way. This included to slim down the standalone so it only runs the exact procedures needed and thus takes less time while executing the same packing orders and reaching the same solutions. Such changes resulted me to re-run the same experiments to have the results and running time if concerned generated under the final version of Cargo Manager.

4.4 Investigating Cargo Manager

As we have introduced Cargo Manager as much as we know in the previous section, there are many aspects remaining unclear and uncontrollable to us. Therefore, we decided to

investigate Cargo Manager (CM) so that we can draw some implications for our algorithms' design. The first aspect we want to know is that how CM would react to different settings in the respect of its execution time. CM's execution time can not be neglected because we will have to run CM every time we generate an actual packing. Therefore, we are interested in investigating the impact of three aspects, the number of boxes (CMI1), the number of box types (CMI2) and container sizes (CMI3) on CM's execution time. We give each investigation its code in brackets so they can be easily referred later. Also, we are interested to investigate the impact of sequencing on the results. Although we have little control over Cargo Manager, we do have control over our inputs towards the CM and sequencing of the box types is one of them. At last, we are not satisfied with the brief description on those constructive heuristic methods GOAL gives us. We would like to run those heuristics separately and compare the results to have a general understanding of which heuristic(s) generally performs the best.

In short, we will carry out three experiments on CM. First experiment is to investigate the impact of the number of boxes (CMI1), the number of box types (CMI2) and container sizes (CMI3) on Cargo Manager's execution time. Second experiment is to investigate the impact of sequencing on the packing results. The last experiment is to investigate the impact of CM's constructive heuristic methods. We also generate our own data sets for our experiments as the ones in the literature are not particularly suitable for the multiple containers problem we are solving. In section 4.4.1, we generate data sets for the investigation on Cargo Manager's execution time only. In section 4.4.2, we present our new benchmark data sets for multi-container packing problem. New benchmark data sets are used in the experiments on the impact of sequencing and heuristics, as well as to generate results for our algorithms in chapter 5 and 6.

4.4.1 Experimental Data for The Investigation on Cargo Manager's Execution Time

As discussed before, we will investigate on the impact of three factors on Cargo Manager's execution time. The three factors are number of boxes (CMI1), number of box types (CMI2) and container sizes (CMI3). Therefore, we need data sets specially designed for each investigation. Each of the data sets has 10 instances, and all boxes are selected from a range of 358 different types of electronic products such as cookers, fridges, fans and the like, i.e. the mother data sample pool of data set soton1. For each instance, all box types are randomly selected before a single box is added from the selected box types each time until the required

		CMI1 (The Impact of box number)												CMI2 (The Impact of Box Type Number)																			
		b5				b10				b25				n100				n250				n500											
		n50	n100	n250	n500	n50	n100	n250	n500	n50	n100	n250	n500	b3	b5	b10	b25	b3	b5	b10	b25	b3	b5	b10	b25								
Used in CMI3 (The Impact of Container Sizes)										X				X												X				X			
Number of Box Types (b)	3	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X								
	5																																
	10																																
	25																																
Number of Boxes (n)	50	X				X				X																							
	100		X				X				X																						
	250			X				X				X																					
	500				X				X				X									X	X	X	X								

Table 4.1 Data Sets for Cargo Manager Investigation

box number is reached. All instances have the standard container size of 12035 * 2350 * 2393(mm).

Overall, we create 24 data sets for these experiments. For investigation CMI1, we generate 12 data sets with all of the instances having same number of box types. These data sets are categorized into three groups with 4 data sets in each group. Data sets in the first group (CMI1b5) all have 5 box types with box number of 50, 100, 250, and 500 respectively. Data sets in the second group (CMI1b10) all contain 10 box types and the data sets in the third group (CMI1b25) all contain 25 box types. The other aspects of the data sets are the same as the ones in the first group. For investigation CMI2, we generate 12 data sets with all of the instances having same number of boxes. Again, three groups are structured for these data sets with 4 data sets in each group. The first group (CMI2n100) contains data sets which all have 100 boxes with box types of 3, 5, 10, and 25 respectively. The second (CMI2n250) and third (CMI2n500) group of data sets are similar compare to the first group only with different box number of 250 and 500 respectively.

For the investigation on the impact of container size (CMI3), only four data sets of the 24 are used. They are data sets with 25 box types and 250 boxes (CMI1b25n250), 25 box types and 500 boxes (CMI1b25n500), 500 boxes and 10 box types (CMI2n500b10), 500 boxes and 25 box types (CMI2n500b25). Nine different container sizes are created. The idea behind this design is to change all three dimensions by the same proportion so that the total container volume is increased or decreased steadily. Taken the standard container volume of V_C , the eight variations of container volume are $0.125V_C$ with each dimension D shorten by 0.5 times, $0.25V_C$ ($0.63D$), $0.5V_C$ ($0.7937D$), $0.75V_C$ ($0.90856D$), $2V_C$ ($1.2599D$), $4V_C$ ($1.5874D$), $6V_C$ ($1.8171D$), and $8V_C$ ($2D$). Table 4.1 summaries the 24 data sets we generated for Cargo Manager Investigation, and the 9 different container sizes used for CMI3 is summarised in table 4.2.

	C0.125	C0.25	C0.5	C0.75	C1 (12035 * 2350 * 2393mm)	C2	C4	C6	C8
Volume	0.125	0.25	0.5	0.75	1	2	4	6	8
Each Dimension	0.5	0.63	0.7937	0.9085	1	1.2599	1.5874	1.8171	2

Table 4.2 Container Sizes for CMI3

4.4.2 New Benchmark Data Generation

The two already existed benchmark data sets designed for multi-container packing problem are Ivancic et al., (1989) and Martello et al., (2000). Although those two data sets are still valid data set for theoretical models, they are not particular suitable for our algorithms. Ivancic et al., (1989) has large item sizes compared to bin size in most of its instances. Therefore, they only reflect a tiny part of the real life container loading scenario. Martello et al., (2000) has all its items rotatable in all instances, and all its instances have highly heterogeneous setting, which is less common in real life problems. Moreover, container packing problem like ours faces more realistic measures such as those described in section 4.2 compared to general bin packing problem in literature. Therefore, we generate new benchmark data sets sampled from two real-industry data sets provided by our sponsor Gower Optimal Algorithms Ltd.

First data s1 samples a range of 358 different types of electronic products such as cookers, fridges, fans and the like from a UK domestic electrical company. The other data s2 provided by a major UK supermarket samples a range of 518 different types of products under from the ‘Paper, Computer and Office Supplies’ category. In general, the box sizes in s2 are smaller than those in s1. Data sets expected to fill approximate 5 and 10 containers are generated separately for s1 and s2 each. Three data sets, intended to represent weakly heterogeneous, mixed, and strongly heterogeneous cases, containing 5, 10 and 25 different box types respectively are generated for both 5-container and 10-container categories. 20 instances are created for each data set. For all data sets, the 40ft Standard Steel Maersk Container with the size of 12035 * 2350 * 2393(mm) is used.

Table 4.3 summarises the structure of our benchmark data sets. Individual data set is given a code based on its compositions. For example, data set sampled from electronic products (s1) aiming to pack 5 containers (c5) with 10 box types (b10) is given code s1c5b10. Data set sampled from the ‘Paper, Computer and Office Supplies’ category (s2) aiming to packing 10 containers (c10) with 25 box types (b25) is given code s2c10b25. Codes are also given to a cluster of data sets. Code s1c5 is given to the cluster of data sets s1c5b5, s1c5b10 and s1c5b25. Codes s1c10, s2c5 and s2c10 are assigned in the same way. Code soton1 is used

		soton1 (s1)						soton2 (s2)					
		s1c5			s1c10			s2c5			s2c10		
		s1c5b5	s1c5b10	s1c5b25	s1c10b5	s1c10b10	s1c10b25	s2c5b5	s2c5b10	s2c5b25	s2c10b5	s2c10b10	s2c10b25
Product Type (s)	Electronic products (1) from the 'Paper, Computer and Office Supplies' category (2)	X	X	X	X	X	X	X	X	X	X	X	X
Estimated Container Number (c)	5 10	X	X	X	X	X	X	X	X	X	X	X	X
Number of Box Types (b)	5 10 25	X	X	X	X	X	X	X	X	X	X	X	X

Table 4.3 Benchmark Data Sets

for all data sets in s1c5 and s1c10, while all data sets in s2c5 and s2c10 are represented as soton2.

The method to generate the data sets is an adaptation of the framework suggested by Bischoff and Ratcliff (1995). In the case of data set of 10 different box types targeting 5 containers sampling from soton1, 10 different box types are randomly selected from 358 possible types, and each box type has the quantity of 1 box. In an effort to make the realistic data, only one dimension is allowed to be placed vertically for our data. However, all boxes can be rotated 90° provided the base stays as the base. Since the new designed data is supposed to create interesting problems, i.e. reducing the number of containers during the process, the total cargo volume is designed to only 85% of the total container volume. The data is generated firstly by comparing the total cargo volume with 85% of the total container volume. If the limit is not reached, a single box of a random box type will then be selected from the already chosen types. The quantity of this box type will thus increase by one. The process is repeated until the 85% limit is reached (or exceeded). The detailed description on generation of a single instance can be found in figure 4.4. Our new benchmark data sets can be downloaded from the ESICUP website (to be uploaded in the future).

Despite the effort to generate sample data set, we think it is worth mentioning that there is no standard type of consignment in real-life industry as the consignment may have as many as 10,000 individual items or as few as a dozen. Regarding the item types, it is hard to say whether weakly or strongly heterogeneous data is more common than the other. It depends on the company and the shipment. They might be all the same but more usually might have a size between 5 to 50 different types.

4.5 Experimental Results

We present results of Cargo Manager Investigation in this section. Firstly, results on Cargo Manager (CM) execution time are present along with discussion on any findings. Then, we present the comparison of results under different sequencings and discuss our findings. Fi-

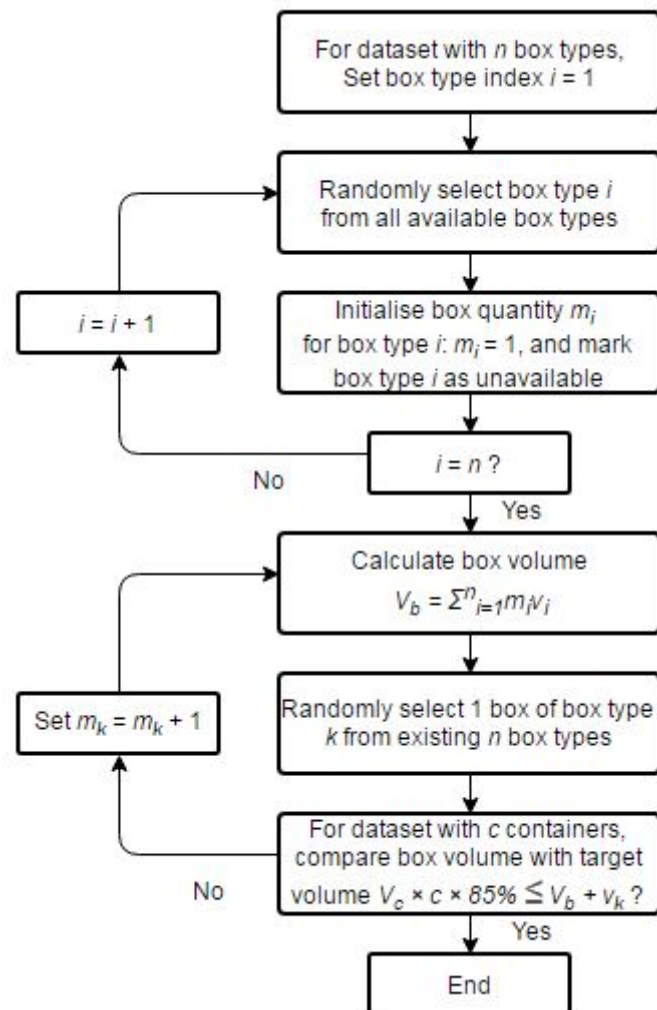


Fig. 4.4 Generation of a single instance

nally, results run using each single CM constructive heuristic are compared and we draw our thought on it. All the procedures involved are coded in Java 1.8.0_05 and all the instances are run on an Intel(R) 2.60 GHz PC with a 4.00 GB RAM.

4.5.1 Cargo Manager Execution Time

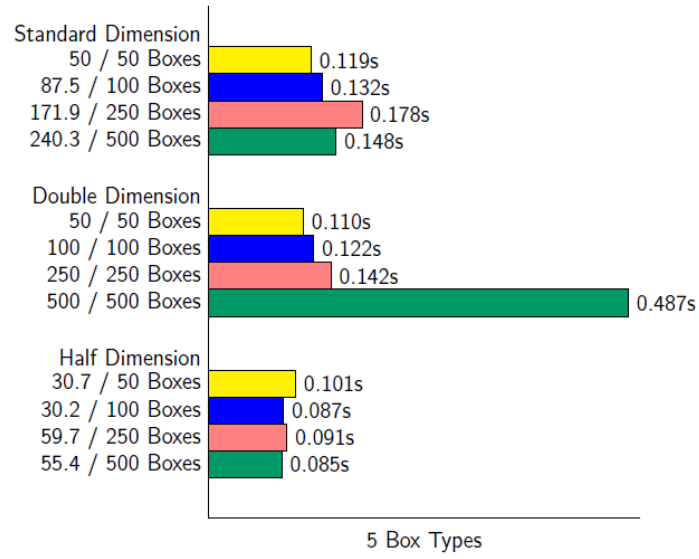
We present our results of Cargo Manager investigation in this section. Execution times are presented for investigations on number of boxes (CMI1), number of box types (CMI2) and container sizes (CMI3). During experiment, Cargo Manager is run once on an instance. The execution time of that running is recorded regardless of the result of packing, i.e. no matter whether all boxes are packed or not. The same instance is run 10 times, and the average of these 10 execution times is taken to represent that instance. These averaged execution times along with number of packed boxes are then being averaged again to present that data set which these 10 instances belong to. When investigating the impact of box type and box number, all instances are run two more times with all three container dimensions doubled and halved respectively.

The results on the impact of box number (CMI1) on CM execution time are presented in figure 4.5. Figure 4.6 summarises the results on the impact of box type (CMI2). On the figures, the number of packed boxes for each data set is shown before the default box types or box numbers for each data set. The execution time is shown in second. Take figure 4.5a as an example, an average of 87.5 boxes are packed for data set with 100 boxes under standard container dimensions. The average execution time is 0.132 second. We can see from the figures that the standard container size is able to completely pack some of the instances. Container with doubled dimensions, which has its volume 8 times of the standard container, is large enough to pack all boxes in any instance. Container with their dimensions halved, which has its volume 0.125 times of the standard container, is small enough so that none instance can be completely packed.

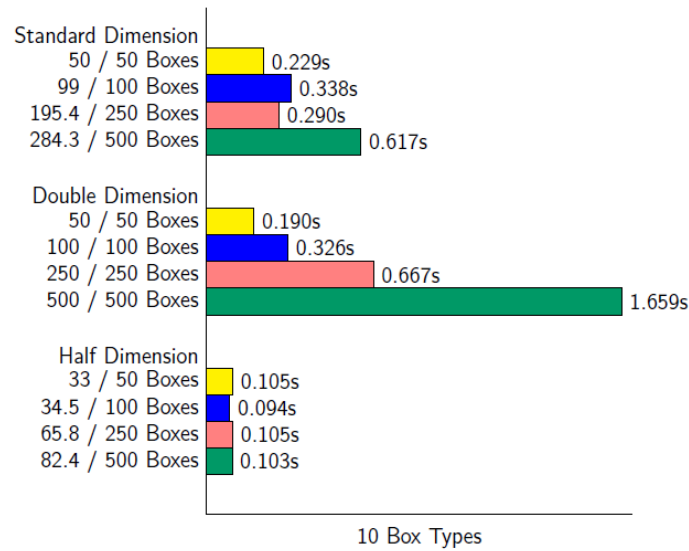
Some common trends are observed after examining all results. Firstly, more boxes result in clearly higher execution time with the same number of box types and the same container dimensions only when the container size is at least large enough, i.e. double container dimension in this case, to pack all the boxes. Clear leaps in execution time are observed when boxes are increased. When container size is not large enough to pack all boxes, i.e. standard container dimension and half container dimension, execution time might only increase slightly even more boxes are actually packed. In almost all cases, execution times remain in a certain range and no obvious increase is observed. On the other hand, more

box types result in higher execution time with same box number and same container dimensions. When container size is larger enough to pack at least half of the boxes, i.e. standard container dimension and double container dimension, clear increase on execution time is observed when box types are increased. Even when container size is small enough to pack only a small portion of the boxes, i.e. half container dimension, increase on execution time can still be observed. Therefore, it seems that box type has a more significant impact on execution time when compared to box number. In general, time used to pack 25 box types increases significantly compared to packing 10 box types with same number of boxes while time used to pack 500 boxes increases in a moderate manner compared to packing 250 boxes with the same box types.

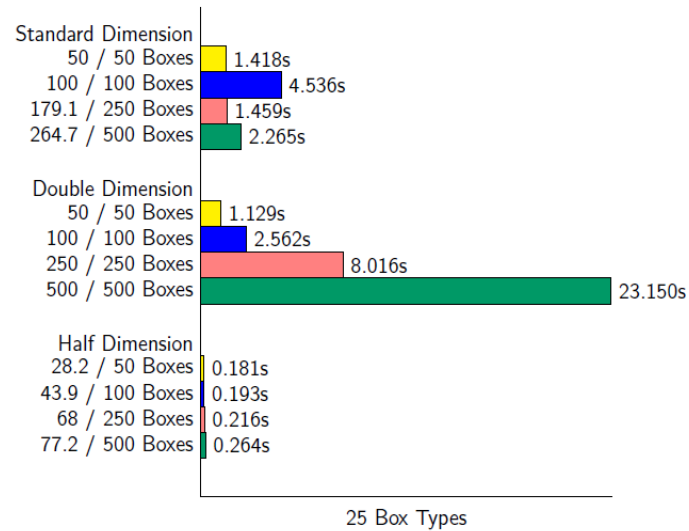
The results on the impact of container size are presented in figure 4.7, 4.8, 4.9 and 4.10. Container size is presented as C0.125 for the smallest container size and C8 for the largest container size. Number of unpacked boxes for each container size is also presented. For data set (CMI1b25n250) with 25 box types and 250 boxes (figure 4.7), almost all boxes are packed under container size of C2 with only 3.9 boxes remaining unpacked. We observe that execution time peaks at 120.095 seconds under container size of C2 from C1's 15.285 seconds. Execution time then slowly decreases to 81.821 seconds once container size is increased to C8. For data set (CMI1b25n500) with 25 box types and 500 boxes (figure 4.8), almost all boxes are packed under container size of C4 with only 3.5 boxes remaining unpacked. Execution time peaks at 284.683 seconds under container size of C4 from C2's 57.298 seconds. It then slowly decreases to 240.135 seconds once container size is increased to C8. For data set (CMI2n500b10) with 500 boxes and 10 box types (figure 4.9), almost all boxes are packed under container size of C4 with 10.3 boxes remaining unpacked. Execution time increases to 19.063 seconds under container size of C4 from C2's 6.14 seconds. It then continues to increase to 24.179 seconds under container size of C6 and decreases to 18.802 seconds once container size is further increased to C8. For data set (CMI2n500b25) with 500 boxes and 25 box types (figure 4.10), almost all boxes are packed under container size of C4 with only 18.5 boxes remaining unpacked. Execution time peaks at 218.288 seconds under container size of C4 from C2's 62.769 seconds. It then slowly decreases to 186.088 seconds once container size is increased to C8. Based on the above observation, we can conclude that execution time is likely to be at its peak when the container size is large enough to pack all the boxes. It then tends to decrease slowly, if not keeping increasing for a short while, when the container size keeps increasing. Last but not the least, it is worth pointing out that while the above observation is the general trend, the actual execution times still differ from instance to instance.



(a) 5 Box Types

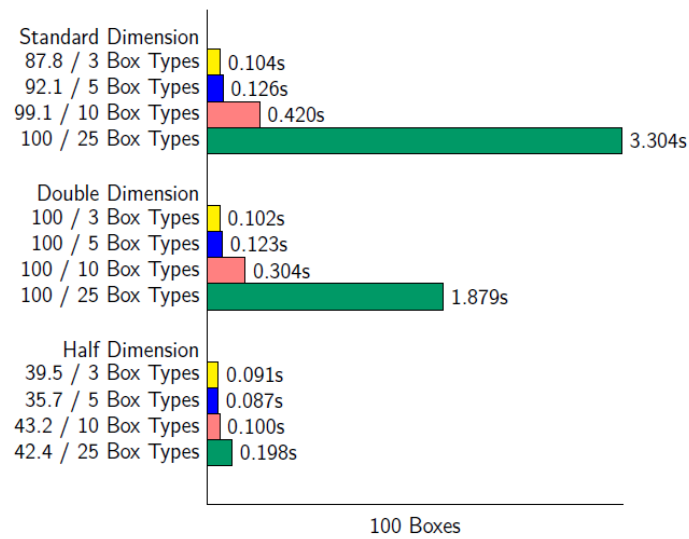


(b) 10 Box Types

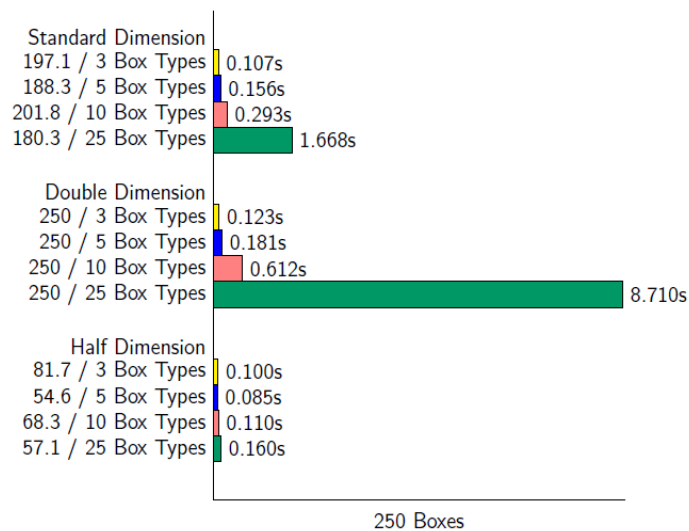


(c) 25 Box Types

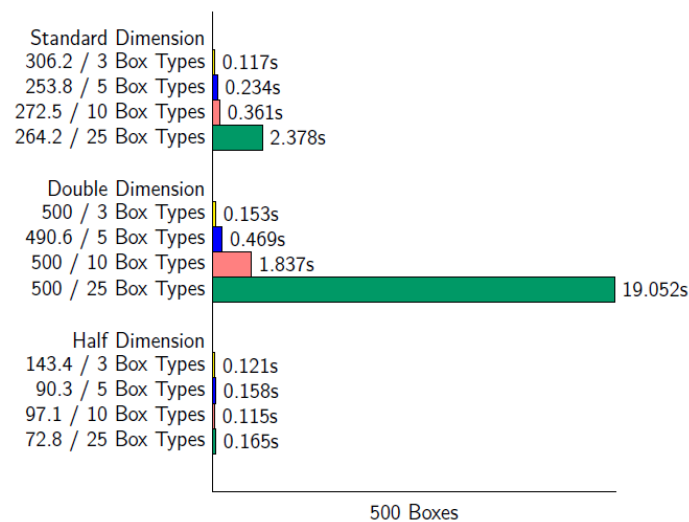
Fig. 4.5 Execution Time for Different Box Types



(a) 100 Boxes



(b) 250 Boxes



(c) 500 Boxes

Fig. 4.6 Execution Time for Different Box Numbers

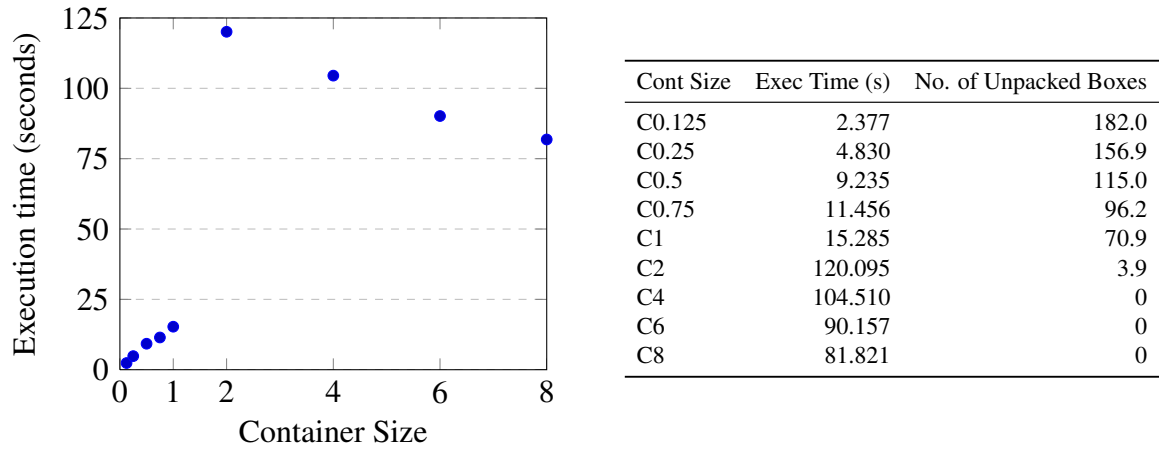


Fig. 4.7 Impact of Container Size (25 Box Types, 250 Boxes)

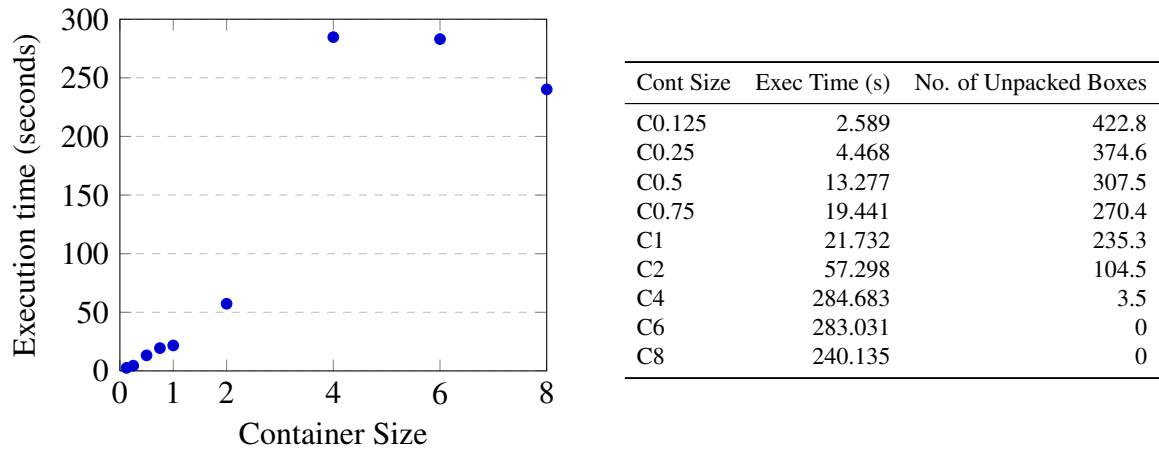


Fig. 4.8 Impact of Container Size (25 Box Types, 500 Boxes)

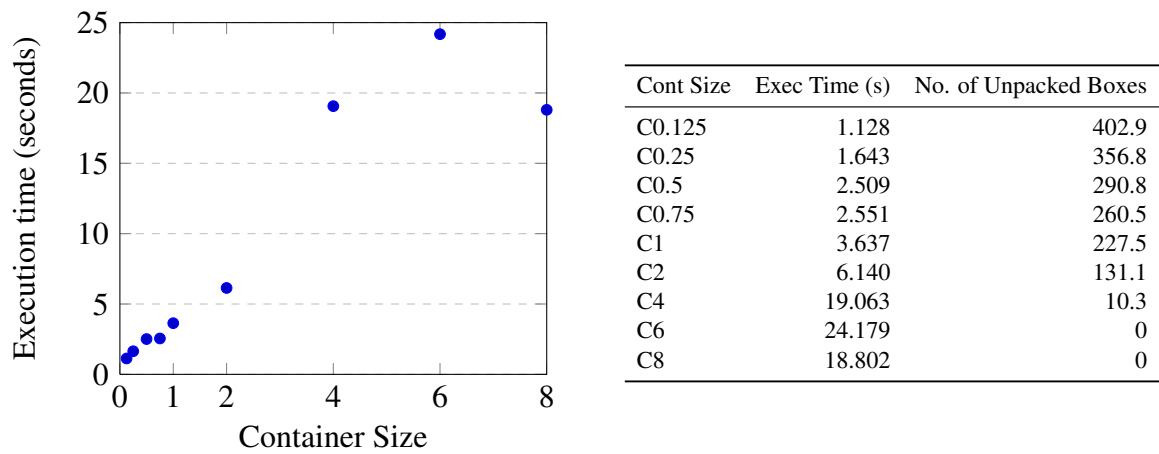


Fig. 4.9 Impact of Container Size (500 Boxes, 10 Box Types)

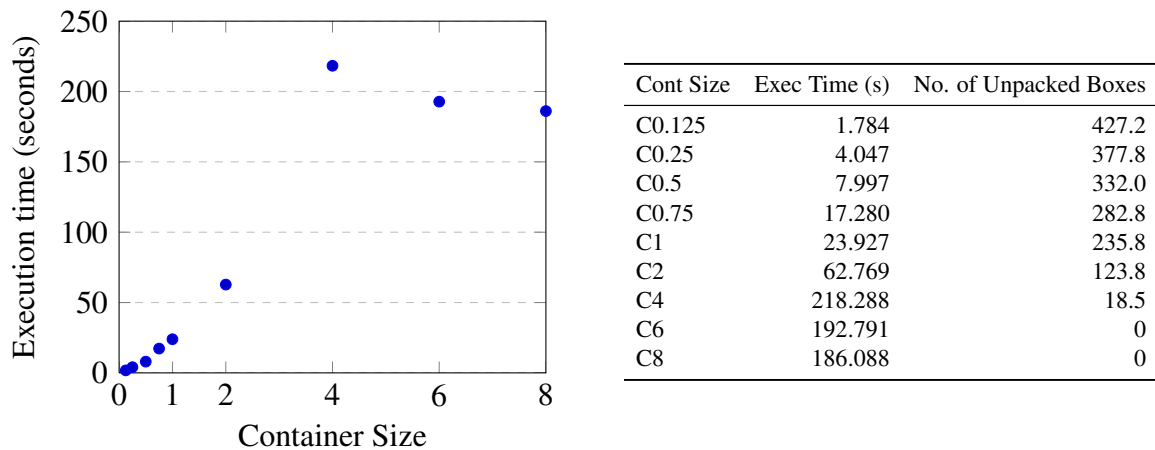


Fig. 4.10 Impact of Container Size (500 Boxes, 25 Box Types)

4.5.2 Impact of Sequencing

In chapter 2, we summarised both the static and dynamic sequencing rules in the literature. Here we choose seven static sequencing rules to test which sequencing rule tends to generate better solutions in general. These rules are number of boxes (N), volume (V), largest dimension (D), largest surface (S), height (H), base area (B), and ratio of base dimensions (R). Among the rules, number of boxes (N) and volume (V) are suitable for situations where rotation of boxes is either allowed or not allowed. Largest dimension (D) and largest surface (S) are used for situation where boxes can be rotated, while height (H), base area (B) and ratio of base dimensions (R) are used for situation where rotation is not allowed. The sequencing rules are summarized in Table 4.4.

Notation	Description	If Used in Rotation
N	No. of Boxes	Both
V	Volume	Both
D	Largest Dimension	Rotation
S	Largest Surface	Rotation
H	Height	No Rotation
B	Base Area	No Rotation
R	Ratio of Base Dimensions	No Rotation

Table 4.4 List of Sequencing Rules

These sequencing rules are combined into sets having a third-tier tie. The box types in the chosen instance are sorted accordingly before GOAL's CM is executed repeatedly till all boxes are packed. Here is an example how it works. In the case of rotation is allowed,

all the sequencing rules which can be used, i.e. number of boxes (N), volume (V), largest dimension (D), and largest surface (S), are formed into combination having a third-tier tie. All the combination are listed below. Take combination NVD as an example. All box types are non-increasingly sorted by number of boxes (N). Box types with ties are then non-increasingly sorted by volume (V). In the case of further ties, box types are non-increasingly sorted based on their largest dimension (D).

- NVD NVS NDV NDS NSV NSD
- VND VNS VDN VDS VSN VSD
- DNV DNS DVN DVS DSN DSV
- SNV SND SVN SVD SDN SDV

Three benchmark data sets from literature are used. BR data set is designed for output maximisation problem by Bischoff and Ratcliff (1995). Among the 15 sets, the first 7 sets are designed for Single Large Object Placement Problem (SLOPP) in which weakly heterogeneous boxes are packed into a single container while the other 8 sets are designed for Single Knapsack Problem (SKP) in which strongly heterogeneous boxes are packed into a single container. Each BR data set has 100 instances. LN data set is designed for SLOPP as well by Loh and Nee (1992). LN data set has 15 instances. IMM dataset is designed for input minimisation problem by Ivancic et al. (1989). The data is designed for Single Stock Size Cutting Stock Problem (SSSCSP) in which weakly heterogeneous boxes are packed into identical containers. IMM data set has 47 instances. We also run experiments on our soton1 and soton2 data sets presented in section 4.4.2.

Table 4.5 and 4.6 summarizes results under different sequencing rules. We report the results in terms of percentage utilisation for data sets BR and LN, and in terms of total number of containers (bins) used for data sets IMM, soton1 and soton2. Data sets BR and LN are designed for packing single container. Therefore, percentage utilisation is calculated as the ratio of total box volume packed in the first container and the volume of first container. Results for 15 BR data sets are separately reported with BR1-15 takes the average of all 15 results. The results shown in the table are the average of all the results with the same first sequencing rule of that data set. As shown in Table 4.5, BR1, BR2, BR3 and BR5 have the best results when largest surface (S) is set as the first sequencing rule. However, volume (V), when set as the first sequencing rule, generates the best results overall with all the rest data sets. For data set LN, volume (V) generates the best percentage utilisation.

Data sets IMM, soton1 and soton2 are designed for packings multiple containers. Their results are presented as the total number of containers used for all instances in each data

set. As presented in Table 4.6, data sets soton1 and soton2 are categorised into 4 groups and a sum is calculated for each group. Because it takes a huge amount of time to run data set soton2 under its usual settings, we adjusted container size under which soton2 runs. Boxes in soton2 are smaller in soton2 compared to those in soton1. Average box size in soton1 is roughly 15 times that of soton2. As our previous experiments show, a high ratio of container size to average box volume and a much larger number of boxes, causes long running time. Therefore, we reduce container size for the same amount by making each container dimension 2.45 times smaller. IMM results show that volume (V) uses the fewest containers. Results of soton1 and soton2 indicate that volume (V) generates the best results in general while height (H) generates some best results in soton1. Thus, we narrow down our first sequencing rule as volume (V).

Dataset	N	V	D	S	H	B	R
BR1	89.283	89.966	89.786	90.331			
BR2	87.952	89.479	89.292	89.652			
BR3	87.354	88.925	88.700	88.960			
BR4	86.498	88.467	88.128	88.358			
BR5	85.449	87.560	87.196	87.624			
BR6	84.754	87.277	86.857	87.205			
BR7	83.099	86.117	85.473	85.778			
BR8	80.927	84.379	83.670	83.919			
BR9	79.094	82.810	81.498	82.279			
BR10	77.725	81.857	80.387	80.850			
BR11	76.820	80.522	79.028	79.124			
BR12	75.931	79.891	77.962	78.629			
BR13	75.355	79.172	77.035	77.550			
BR14	74.258	78.372	76.092	76.372			
BR15	74.159	77.795	75.238	75.550			
BR1-15	81.244	84.173	83.089	83.479			
LN	69.461	69.910			69.460	69.493	69.212
IMM	752.000	732.333	747.000	734.000			

Table 4.5 Results of Different Sequencing Rules for Literature Benchmark Data Sets

We then further investigate the impact of combination of sequencing rules with volume (V) as the first sequencing rule. As shown in Table 4.7, six combinations are available for data sets BR and IMM in which all three dimensions are allowed to be height. The results show no significant difference among different combinations. Table 4.8 presents results for data set LN, soton1 and soton2. Because data sets LN, soton1 and soton2 have 1 dimension fixed as height, there are twelve combinations and all of them give the same result.

Dataset	N	V	H	B	R
s1c5b5	58.25	56	56	62	61
s1c5b10	58	55	56	57	63
s1c5b25	59	54	53.75	55	60
s1c5 Sum	175.25	165	165.75	174	184
s1c10b5	120	114	112	121	119
s1c10b10	114	106	108	113	117
s1c10b25	110.5	108	107	109	113
s1c10 Sum	344.5	328	327	343	349
s2c5b5	821	794	800	800	886
s2c5b10	762.75	710	714.25	718	795
s2c5b25	630	602	608.5	606	642
s2c5 Sum	2213.75	2106	2122.75	2124	2323
s2c10b5	1660.5	1559	1588.25	1560	1701
s2c10b10	1639	1461	1474.5	1493	1676
s2c10b25	1235.25	1184	1194	1195	1252
s2c10 Sum	4534.75	4204	4256.75	4248	4629

Table 4.6 Results of Different Sequencing Rules for Soton Data Sets

We can conclude that setting the right primary sorting criteria is the most important towards obtaining a good performance. As different combinations fail to show significant difference, we consider the best sequencing strategy as VSD (volume, largest surface, largest dimension) for data sets which allow box rotation and VHB (volume, height, base area) for data sets which do not allow rotation. We decide our second and third sequencing rule based on their overall performances in Table 4.5 and 4.6.

4.5.3 Results under Different Constructive Heuristics

Under the best sequencing strategies generated in the previous section, we make CM run a single heuristic only each time rather than running all heuristics and picking the best result. Sequencing strategy VSD is set for data sets BR and IMM, while LN, soton1 and soton2 have VHB as their sequencing. All CM heuristics described in section 4.3.2 are included in the comparison. Table 4.9 and 4.10 shows the full results run by 16 heuristics separately. Among 15 BR data sets, H2 generates 5 best results and H4 generates 4 best results. Overall, H2 generates the best average result on BR data set with H4 result closely follows. Heuristic S2 generates the best result for LN data set, and best result on IMM data set was generated by heuristic S0, S2, S10 and S12. It is worth mentioning that S2 uses the exact same

Dataset	VND	VNS	VDN	VDS	VSN	VSD
BR1	89.966	89.966	89.966	89.966	89.966	89.966
BR2	89.479	89.479	89.479	89.479	89.479	89.479
BR3	88.925	88.925	88.925	88.925	88.925	88.925
BR4	88.467	88.467	88.467	88.467	88.467	88.467
BR5	87.560	87.560	87.560	87.560	87.560	87.560
BR6	87.277	87.277	87.277	87.277	87.277	87.277
BR7	86.117	86.117	86.117	86.117	86.117	86.117
BR8	84.379	84.379	84.379	84.379	84.379	84.379
BR9	82.810	82.810	82.810	82.810	82.810	82.810
BR10	81.857	81.857	81.857	81.857	81.857	81.857
BR11	80.522	80.522	80.522	80.522	80.522	80.522
BR12	79.895	79.895	79.887	79.887	79.890	79.890
BR13	79.173	79.173	79.173	79.173	79.172	79.172
BR14	78.371	78.371	78.374	78.374	78.372	78.372
BR15	77.795	77.789	77.798	77.798	77.791	77.796
BR1-15	84.173	84.173	84.173	84.173	84.172	84.173
IMM	731	731	733	733	733	733

Table 4.7 Results of Different V Sequencing Rules on BR and IMM

heuristic as H2 only with container length and width swapped. For data sets soton1 and soton2, heuristic H0 generates the best results in general with heuristic H2 comes second. For data sets s1c5, s1c10 and s2c5, the results generated by H2 is only slightly behind by 1 container compared to those generated by H0. For all the data sets including those ones from literature, the most complex and time consuming heuristic H3 however does not perform as well as expected.

In summary, Heuristic H0 and H2 generate better results compared to the rest heuristics in general. However, other heuristics may occasionally generate the best results. Excluding them from operation may lead to the risk of losing better solutions. Therefore, users are faced with a dilemma. We may choose to include all or a few heuristics to have a better final result with the cost of a longer running time. The other option is to include only one or minimum heuristics for a reduced running time with the risk of missing better solution. However, the decision for us is much easier due to the nature of this project. As we will build search heuristics with Cargo Manager embedded in. We rely on CM's constructive heuristics to do the actual packing. Therefore, we may allow Cargo Manager to run one heuristic only to ensure the consistence of our packing placement.

Dataset	VNH	VNB	VNR	VHN	VHB	VHR	VBN	VBH	VBR	VRN	VRH	VRB
LN	69.910	69.910	69.910	69.910	69.910	69.910	69.910	69.910	69.910	69.910	69.910	69.910
s1c5b5	56	56	56	56	56	56	56	56	56	56	56	56
s1c5b10	55	55	55	55	55	55	55	55	55	55	55	55
s1c5b25	54	54	54	54	54	54	54	54	54	54	54	54
s1c10b5	114	114	114	114	114	114	114	114	114	114	114	114
s1c10b10	106	106	106	106	106	106	106	106	106	106	106	106
s1c10b25	108	108	108	108	108	108	108	108	108	108	108	108
s2c5b5	794	794	794	794	794	794	794	794	794	794	794	794
s2c5b10	710	710	710	710	710	710	710	710	710	710	710	710
s2c5b25	602	602	602	602	602	602	602	602	602	602	602	602
s2c10b5	1559	1559	1559	1559	1559	1559	1559	1559	1559	1559	1559	1559
s2c10b10	1461	1461	1461	1461	1461	1461	1461	1461	1461	1461	1461	1461
s2c10b25	1184	1184	1184	1184	1184	1184	1184	1184	1184	1184	1184	1184

Table 4.8 Results of Different V Sequencing Rules on LN and soton

Dataset	H0	H2	H3	H4	H5	H10	H12	H13	S0	S2	S3	S4	S5	S10	S12	S13
BR1	88.644	88.803	88.629	88.509	87.977	88.363	88.486	88.209	89.150	89.188	89.103	89.134	88.945	88.900	88.924	88.888
BR2	87.561	87.599	87.550	87.528	86.686	87.123	87.186	86.934	89.167	89.155	89.129	89.111	89.098	89.085	89.065	89.066
BR3	86.312	86.073	86.187	86.246	84.834	85.310	85.461	85.281	88.512	88.472	88.477	88.501	88.425	88.438	88.438	88.463
BR4	85.366	85.560	85.171	85.033	84.624	84.913	84.927	85.015	88.143	88.157	88.144	88.196	88.100	88.109	88.109	88.108
BR5	87.301	87.326	87.331	87.351	87.157	87.120	87.211	87.198	84.149	84.371	84.330	84.669	80.173	80.686	80.857	80.649
BR6	87.130	87.094	87.130	87.090	87.032	86.876	86.875	86.910	83.806	83.946	83.961	83.987	79.308	78.552	78.951	78.739
BR7	85.893	85.952	86.001	85.992	85.817	85.648	85.682	85.660	82.530	82.643	82.617	82.706	77.874	75.917	76.134	76.103
BR8	84.207	84.209	84.148	84.173	83.638	83.435	83.387	83.390	81.225	81.300	81.223	81.397	77.361	72.502	72.938	72.985
BR9	82.447	82.539	82.423	82.413	81.470	80.943	80.851	80.872	79.679	79.705	79.638	80.176	76.451	69.836	70.103	70.232
BR10	81.310	81.407	81.358	81.537	79.982	78.656	78.642	78.683	78.754	79.024	79.081	79.383	75.827	68.530	69.049	68.905
BR11	79.763	79.865	79.797	79.941	78.377	76.754	76.657	76.686	77.590	77.661	77.711	78.214	74.962	67.343	67.463	67.451
BR12	78.976	79.058	79.020	79.500	77.685	75.846	76.120	76.140	76.677	76.792	76.737	77.401	74.513	66.815	67.013	67.039
BR13	78.349	78.465	78.377	78.354	77.284	75.092	74.778	74.711	76.211	76.343	76.302	76.663	74.499	65.952	66.214	66.229
BR14	77.263	77.397	77.378	77.337	76.060	73.789	74.047	74.026	74.984	75.053	75.035	75.722	73.425	65.182	65.447	65.304
BR15	76.720	76.931	76.903	76.732	76.103	73.750	73.465	73.438	74.571	74.727	74.718	75.065	73.407	65.124	65.415	65.434
BR1-15	83.149	83.219	83.160	83.182	82.315	81.575	81.585	81.544	81.677	81.769	81.747	82.022	79.491	75.398	75.608	75.573
LN	69.252	69.293	69.136	69.169	67.746	68.693	68.109	67.521	69.549	69.685	69.162	69.304	68.457	68.195	68.215	68.188
IMM	750	750	751	748	784	750	750	751	735	735	737	738	769	735	735	737

Table 4.9 Results of Different Heuristics for Literature Benchmark Data Sets

Dataset	H0	H2	H3	H4	H5	H10	H12	H13	S0	S2	S3	S4	S5	S10	S12	S13
s1c5b5	57	56	56	56	55	58	56	59	57	58	58	58	58	57	57	58
s1c5b10	56	56	58	57	56	58	57	59	56	56	56	56	56	57	58	57
s1c5b25	54	56	56	57	58	58	58	59	54	55	55	56	60	59	59	59
s1c10 Sum	167	168	170	170	169	174	171	177	167	169	169	170	174	173	174	174
s1c10b5	115	115	115	115	115	116	115	116	114	116	116	117	117	116	116	118
s1c10b10	110	111	112	112	111	110	111	113	110	112	112	112	112	110	111	112
s1c10b25	109	109	109	109	111	109	110	109	110	109	109	110	111	112	112	111
s1c10 Sum	334	335	336	336	337	335	336	338	334	337	337	339	340	338	339	341
s2c5b5	792	791	811	808	792	795	796	828	794	801	803	804	801	795	806	817
s2c5b10	711	712	726	728	712	716	715	751	711	717	716	717	715	712	717	724
s2c5b25	609	610	615	613	612	616	617	638	611	614	614	614	617	612	615	618
s2c5 Sum	2112	2113	2152	2149	2116	2127	2128	2217	2116	2132	2133	2135	2133	2119	2138	2159
s2c10b5	1567	1575	1595	1597	1575	1578	1583	1657	1568	1570	1571	1578	1571	1571	1572	1580
s2c10b10	1462	1462	1482	1480	1463	1477	1476	1539	1470	1469	1466	1469	1469	1474	1472	1482
s2c10b25	1185	1193	1203	1208	1197	1196	1200	1240	1184	1193	1191	1191	1194	1188	1195	1201
s2c10 Sum	4214	4230	4280	4285	4235	4251	4259	4436	4222	4232	4228	4238	4234	4233	4239	4263

Table 4.10 Results of Different Heuristics for Soton Data Sets

4.6 Conclusion

In this chapter, we introduced our sponsor and the unique request they presented to us. We explained Cargo Manager as a standalone software in as much detail as we could. The focus is on our investigation of Cargo Manager. We investigated the impact of the number of boxes, the number of box types and container sizes on Cargo Manager's execution time. We found out that compared to the number of boxes, the number of box types has a more significant impact on the execution time. The increasing of execution time is not proportional to the increasing of the number of box types. A few extra box types may result in a much longer execution time. Moreover, for the same box sets a larger container size results in a longer execution time. The execution time usually is at its peak when the container is just larger enough to pack almost all boxes. Also, we investigated the impact of sequencing and heuristics on packing results. The best 3-tier sequencing strategy for data sets which allow box rotation is volume, largest surface and largest dimension (VSD), while for data sets which do not allow rotation volume, height and base area (VHB) serves as the best strategy. On the heuristic side, two simple wall building heuristics H0 and H2 generate the best results in most cases.

These findings are valuable as they provide us implications for our algorithm design in chapter 5 and 6. The experiments on Cargo Manager's execution time confirmed the high cost of CM's running time especially when there is a large number of box types or/and the container size is much larger compared to box size, i.e. this would be the case of data set soton2 under 40ft container. Cargo Manager is the only mean to evaluate a packing and is time consuming, while we have much exploration to do within a neighbourhood and in different neighbourhoods. Therefore, it is important to design a search with the right balance of calling CM and exploring our search neighbourhood. The findings on the best performing sequencings and heuristics are directly contributed to the experimental runs of our algorithms in chapter 5 and 6. A selected sequencing and a constructive heuristic are implemented throughout all the experimental runs of our algorithms. We also generate two new benchmark data sets for multi-container packing problem, as we feel that the current data sets in the literature fall short to satisfy the modern day needs.

Chapter 5

Homogeneous Container Problem

5.1 Problem Description

The homogeneous container problem we are interested in solving here is an input minimisation problem defined as follows. There is an unlimited number of rectangular containers of identical dimensions and cost. A given number of rectangular boxes are categorised into N types. All box types have different dimensions, and boxes of each type are all unique. Each box type is represented by B_1, B_2, \dots, B_N . There are n_j available boxes of type j where $1 \leq j \leq N$. The objective is to pack all boxes into as few identical containers as possible.

Because the number of containers used is too coarse a measurement to sufficiently discriminate between solutions, we consider overall utilisation across all used containers will reflect more about the quality of a solution. Therefore, we evaluate solution quality through overall utilisation which is to be maximised. We consider the boxes in our benchmark data sets *soton1* and *soton2* as weakly heterogeneous boxes. Therefore, we are solving Single Stock-Size Cutting Stock Problem (SSSCSP) where weakly heterogeneous boxes are packed into identical containers. This aspect will only be reflected in our experiments in section 5.6. All the algorithms we design in this chapter are suitable to run data sets with strongly heterogeneous boxes. As our algorithms are heuristics, all solutions obtained are approximate.

The structure of this chapter is as follows. In section 5.2, we introduce our iterated local search with intelligent neighbourhood (ILSIN). We then explain iterated local search with descent and randomness (ILSDR) in section 5.3 and iterated local search with simple neighbourhood (ILSSN) in section 5.4. A completely different approach, beam search, is

introduced in section 5.5. At last, computational results are reported and comparison is drawn before a short conclusion ending the chapter.

5.2 Iterated Local Search with Intelligent Neighborhood (ILSIN)

5.2.1 Notation

Before introducing our algorithm in detail, we would like to list all the notations used in Iterated Local Search with Intelligent Neighbourhood (ILSIN).

- R be a fixed sequencing rule
- LB_{Awk} be the lower bound of box number selected for type t
- UB_{Awk} be the upper bound of box number selected for type t
- L be the length of container
- W be the width of container
- H be the height of container
- V_c be the volume of a single container
- E be the number of containers currently used
- C_e be the packed container C_e , where $e = 1, 2, \dots, E$
- L_m be the least container length used among E packed containers
- L_m^* be the least container length used among E packed containers under the best solution
- V_B be the total box volume
- T be the total number of box types
- t be the box type t , where $t = 1, 2, \dots, T$
- l_t be the length of box type t
- w_t be the width of box type t

- h_t be the length of box type t
- m_t be the total box number of box type t
- J be the total number of blocks found among E containers
- B_j be the block B_j separated from container(s), where $j = 1, 2, \dots, J$
- L_j be the length of block B_j
- U_j be the utilisation of block B_j
- V_j be the packed box volume of block B_j
- U_m be the user-defined utilisation deciding if a block B_j is to be unpacked
- V_U be the total box volume unpacked from block(s)
- P_e be the consolidated empty block in the packed container C_e
- V_e be the total box volume previously packed within the consolidated empty block P_e
- L_e be the length of the consolidated empty block P_e
- U_e be the utilisation before the consolidated empty block P_e being unpacked, i.e. $U_e = \frac{V_e}{L_e \times W \times H}$
- U_H be the highest utilisation among all U_e
- F be a list where all unpacked boxes are stored to
- V_F be the total volume of unpacked boxes in list F
- V_F^* be the total volume of unpacked boxes in list F under the best solution
- I be the number of box types in list F
- i be the box type i in list F , where $i = 1, 2, \dots, I$
- n_i be the current available box number of type i
- h_i be the height of box type i
- k_i be the maximum box number of type i stacking onto each other within block height H
- r be the box type r in list F , where $r \in I \& r \neq i$
- h_r be the height of box type r

- k_r be the maximum box number of type r stacked onto a selected number of type i within block height H
- U_c be the current utilisation after packing the current consolidated empty block P_e
- U_b be the best utilisation among all U_c

5.2.2 Design and Framework

With the relationship between permutation and packing remaining not straightforward and having control only on the input, we choose iterated local search because its simple and few parameters characteristic fits well with the nature of the problem we deal with. Since we do not have control over the constructive heuristic itself, it is pointless to change parameter blindly just to get a better result while having no knowledge about why certain parameters generate better results than others. The other reason we choose iterated local search is that the modular approach of ILS allows us to have the flexibility of changing steps if the current framework does not perform. The result of experiments on the Cargo Manager execution time, which is shown in section 4.5.1, suggests that executing Cargo Manager can be very costly.

In this model we decide to design a framework in which we can exploit problem specific knowledge and generate more intelligent neighbourhood moves. The key thing is to evaluate fewer moves but more promising ones within the same neighbourhood. Also, extra time may be saved for exploring towards other neighborhoods. Along with this main goal in mind, we implement some small designs in the effort to compensate our lack of control over the CM heuristics.

Firstly, we realize that CM will always generate a greedy solution based on the given box type sequencing. A greedy solution in a container might result in bad packing in later containers, and the bad packing might outweigh the benefit brought by the earlier greedy packing. Therefore, we suggest an alternative way which generates a less greedy initial solution by semi-randomly pre-assigning boxes into containers before actually packing them.

Secondly, we realize the limitation that upon execution, Cargo Manager packs (and repacks) the whole container including those sections which may have already packed well. Repacking them may be a waste of time and may lead to worse results. Therefore, instead of repacking the whole container, we partition the container so that repacking now only happens within certain sections of the container. Hence well packed section of container do not have to be unpacked every single time.

In addition, since Cargo Manager tends to pack all identical boxes together we try to split the same type of boxes into smaller batches by defining them as more types. The split is done by re-defining a box type as different types. On the input file, a single box type may be split into two or more batches. We hope this move might lead us to different solution spaces because CM may pack the container in a completely different way based on the numbers of new types and their positions in the sequences.

The framework of ILSIN is presented in Algorithm 1. As shown in the algorithm, ILSIN can be separated into five stages. The algorithm firstly generates a partial initial solution. The algorithm then forms a new neighbourhood around the current (partial) solution. Local optimal is therefore being searched within that neighbourhood. Once improvement is spotted, a new neighbourhood is formed around the updated (partial) solution. Upon reaching an approximate local optimal, a kick phase is implemented to kick the current (partial) solution into a completely different space. A final complete solution is generated when time limit is reached.

Algorithm 1 Iterated Local Search with Intelligent Neighborhood (ILSIN)

```

1: Generate Partial Initial Solution
2: while time limit is not reached do
3:   while improvement is found at least once do
4:     Form a New Neighbourhood - Algorithm 1a
5:     Search within the Neighbourhood - Algorithm 1b
6:   end while
7:   Kick Phase
8: end while
9: Generate Final Solution

```

5.2.3 Generate Partial Initial Solution

We propose two choices of generating partial initial solution, by using a fixed sequencing rule R or by using the pre-assignment method. Using a fixed sequencing rule denoted by R is one of the two ways we suggest to generate partial initial solution. By taking such approach, the majority of the boxes are packed within the partial initial solution with only a fraction of the boxes remaining unpacked. We then improve the overall utilisation across these E containers by re-arranging boxes among as well as packing the remaining unpacked boxes into these E containers.

All box types are sorted into a certain order using R onto the input file shown in figure 4.3. The fixed sequencing rule R is a user defined parameter which is decided through the ex-

periment presented in section 4.5.2, and the exact R we use can be found in section 5.6.1. Cargo Manager is then used to pack E containers one by one. E is also a user defined parameter and can be decided by any rule set by the user. One way to decide the value of E is to calculate the ratio of 85% of total box volume V_B and single container volume V_c and take the integer part of the ratio as E . It is possible to define this relationship as

$$E \leq \frac{85\% \times V_B}{V_c} < E + 1.$$

85% is an acceptable starting point for decent packing result. By deciding E in such a way, the majority of the boxes can be packed while hopefully only a few are left unpacked. Among the 16 separate heuristics available in Cargo Manager, only one heuristic is chosen. The decision is made through the experiment presented in section 4.5.3, and the exact heuristic we use can be found in section 5.6.1. Any boxes remaining unpacked are stored into a list F , and in the case of all boxes packed list F stays empty.

Other than generating a partial initial solution by sequentially packing E containers using a fixed sequencing rule R , we alternatively propose a pre-assignment strategy which packs E containers simultaneously. The purpose of this design is to avoid packing the first few containers too greedily and leave awkward boxes to the end. Also, we try to distribute boxes of the same type over the containers as evenly as possible, so they have more combination options to choose from rather than always packing the same type of boxes together. A box type t is defined as awkward if

$$\frac{H}{h_t} < 2, \frac{W}{w_t} < 2 \text{ or } \frac{L}{l_t} < 2.$$

All awkward box types are categorised in Set A, and the remaining box types are categorised in Set B.

We firstly allocate Set A boxes across containers. For awkward box type t , we select a random number of boxes within the range of upper bound UB_{Awk} and lower bound LB_{Awk} . If the box type t with a total number of m_t boxes satisfies the following requirement

$$\frac{L \times W \times H}{l_t \times w_t \times h_t \times m_t} \leq 1,$$

then

$$UB_{Awk} = \frac{m_t}{\frac{E}{2}} \text{ and } LB_{Awk} = \frac{m_t}{E}.$$

If the box type t satisfies

$$\frac{L \times W \times H}{l_t \times w_t \times h_t \times m_t} > 1,$$

then

$$UB_{Awk} = m_t \text{ and } LB_{Awk} = \frac{m_t}{2}.$$

The reason for setting up UB_{Awk} and LB_{Awk} this way is to make sure boxes of same type spread throughout different containers but not in a too heterogeneous way. If the current box number left is no more than LB_{Awk} , all boxes of this type are selected.

Once the number of to-be-assigned boxes is decided, we select a random container from those currently available ones. If the remaining space of the selected container is not enough for the selected boxes, a second chance is given if the remaining space of the selected container is enough for two-thirds of the selected boxes. If the requirement for second chance is still not achieved, the current selected container is marked as unavailable for this box type and another random container is selected. The selection of random container continues till selected boxes are assigned to the successful container. The successfully assigned container is then marked as unavailable for this box type, and the above process continues till either all boxes in this type are assigned or no available container exists for this box type. We reset all containers to be available again before moving on to the next box type.

Once we assign all the awkward box types, the rest of the box types, i.e. Set B, are assigned using the same method. The reason to assign awkward box types first is because those boxes are considered difficult to pack. Therefore, we make sure we assign all of them first and hopefully the other boxes can be fit into the remaining space. Unassigned boxes are stored in the list F under the same fixed sequencing rule R . Cargo Manager is then used to pack each container with its pre-assigned boxes sorted by box type under rule R . In case of boxes unable to be packed, those unpacked boxes are also added to the list F .

5.2.4 Form a New Neighbourhood

As demonstrated in algorithm 1a, to form a new neighbourhood separate blocks among all containers are firstly identified and the best solution is updated. A selected number of blocks are then to be unpacked, following the consolidation of emptied blocks within each container.

Since the CM heuristic works in such a way that creates completely separate blocks along the container length, i.e. no overhang between two blocks, we read the coordinate file gen-

Algorithm 1a ILSIN: Form a New Neighbourhood

Set $V_F^* = V_B$ & $L_m^* = L$ if this algorithm is called first time

Phase 1 - Identify Independent Blocks B_j

```

1: for each container  $C_e$ , where  $e = 1, 2, \dots, E$  do
2:   if  $B_j$  is identified then
3:      $J = J + 1$  & record  $U_j$ 
4:   end if
5: end for

```

Phase 2 - Update Best Solution

```

6: Calculate the volume of all unpacked boxes  $V_F$  in list  $F$  and the least length used among containers  $L_m$ 
7: if  $V_F < V_F^*$  then
8:    $V_F^* = V_F$  &  $L_m^* = L_m$ 
9: else if  $V_F = V_F^*$  then
10:  if  $L_m < L_m^*$  then
11:     $V_F^* = V_F$  &  $L_m^* = L_m$ 
12:  end if
13: end if

```

Phase 3 - Unpack Block(s) B_j with $U_j < U_m$

```

14: Set  $V_U = 0$ 
15: for blocks  $B_j$  with  $U_j \neq 0$ , where  $j = 1, 2, \dots, J$  do
16:   Randomly select a block  $B_j$ 
17:   if  $U_j < U_m$  then
18:      $V_U = V_U + V_j$ 
19:     if  $V_U \geq 10\% \cdot V_B$  &  $V_U \leq 25\% \cdot V_B$  then
20:       Unpack  $B_j$  & End Phase 3
21:     else if  $V_U < 10\% \cdot V_B$  then
22:       Unpack  $B_j$ 
23:     else if  $V_U > 25\% \cdot V_B$  then
24:        $V_U = V_U - V_j$ 
25:     end if
26:   end if
27: end for
28: if  $V_U < 10\% \cdot V_B$  then
29:    $U_m = U_m + 5\%$  & Return to Step 15
30: end if

```

Phase 4 - Consolidate Emptied Blocks

```

31: Set  $U_H = 0$ 
32: for each container  $C_e$ , where  $e = 1, 2, \dots, E$  do
33:   Consolidate emptied blocks to form an consolidated empty block  $P_e$  with the length of  $L_e$ 
34:   Obtain the previously packed total box volume  $V_e$ , and calculate  $U_e = \frac{V_e}{L_e \times W \times H}$ 
35:   if  $U_e > U_H$  then
36:      $U_H = U_e$ 
37:   end if
38: end for

```

erated by Cargo Manager to identify all such blocks. Occasionally no separate block may be observed, we treat all packed boxes in the container as a whole block. These blocks all have width W and height H same as container but different length L_j . Because there is no gap between two completely separate blocks, the block length L_j of block B_j is always the same as the maximum length used in that block. For the last block packed in the container where full container length is not used, an extra empty block usually with a tiny length is created. We then calculate the utilisation of each block. We also record the total volume V_F of boxes which are not packed, i.e. all those boxes in the list F , as well as the least container length used L_m among all E packed containers.

Next, we compare the current volume V_F of remaining boxes, i.e. all those boxes in the list F , with the best one in record V_F' . If the current volume of remaining boxes is lower than the current best one, we accept the current result as the best. In the case of a volume tie, we compare the least length used among containers. The solution with the shorter length becomes the best. The volume of remaining boxes under partial initial solution is automatically recorded as the best at the beginning.

We then randomly loop through all J existing blocks and unpack those with utilisation U_j lower than U_m . U_m is a user-defined parameter, and the value of U_m we use can be found in section 5.6.1. An upper bound of 25% and a lower bound of 10% of total box volume are put on the unpacked box volume. These two bounds are set to make sure that we do not unpack too many boxes making it a complete repacking but still do unpack enough boxes. If the next unpack exceeds the upper bound, the selected block will be dropped and another block will be randomly selected within the allowed range. The unpacking process stops when the current unpacked volume has already exceeded the lower bound. In the case of failing to reach the lower bound, A 5% increase is added to the current unpack cap U_m until the lower bound is reached.

The remaining blocks within their own container are consolidated as well as the emptied blocks, and this results in one empty block in each container only. The utilisation of each consolidated empty block before unpack is compared, and the highest one is noted as U_H . The unpacked boxes are restored to the list F .

5.2.5 Search within the Neighbourhood

We loop through all consolidated empty blocks P_e following the procedures described in algorithm 1b. When searching within the neighbourhood, the main procedures are to firstly

evaluate box type combinations, to pack consolidate empty block based on the best combinations, to split original box type of the best packing to further explore solution space, and to accept or reject solution based on the condition of improvement.

Algorithm 1b ILSIN: Search within the Neighbourhood

```

1: for each consolidated empty block  $P_e$  where  $e = 1, 2, \dots, E$  do
2:   Evaluate available combinations of any two box types, and record those well-fit ones
   within the current block  $P_e$ 
3:   for each box type  $i$  in list  $F$  where  $i = 1, 2, \dots, \frac{I}{2}$  and  $i$  represents the sequencing
   position of  $i$ th do
4:     Mark box type  $i$  as the main box type, and make it the 1st box type of list  $F$ 
5:     Mark box types that fit well with the main box type as sub box types, and relo-
   cate them right after the main box type
6:     Call Cargo Manager to pack current block  $P_e$ , and record current utilisation  $U_c$ 
7:     Compare  $U_c$  with  $U_b$ , and update  $U_b = U_c$  if  $U_c > U_b$ 
8:     Move box type  $i$  down 1 position, and call CM each time until box type  $i$  is
   below all of its sub box types
9:     Restore list  $F$ 's sequencing to the original order before Step 3, and move box
   type  $i$  to the end of the list
10:  end for
11:  Split each original box type from the sequencing order which generates the best
   utilisation  $U_b$  into 2, 3 and 4 batches (i.e. new box types with exact same box di-
   mensions), and call Cargo Manager each time
12:  Compare  $U_b$  with the utilisations generated by 3 splits, and update  $U_b$  accordingly
13:  if  $U_b > U_H$  or improvement is made on any early block then
14:    Accept solution
15:  else
16:    Reject solution, and move current block  $P_e$  to the end of the block list
17:  end if
18: end for

```

Before packing each block using the available boxes in list F , we evaluate box combination in pairs to find those that fit together well along the height and width. For each available box type i where $i = 1, 2, \dots, I$, we find out the maximum number k_i of the same type of boxes it can stack onto each other given its box height h_i and block height H where

$$k_i \leq \frac{H}{h_i} < k_i + 1.$$

If the current available box number n_i is fewer than the theoretical maximum number k_i , we choose the available box number as the maximum stack number.

We then loop through from 1 box to k_i number of boxes for the following steps. For 1 box

of type i , we loop through the rest of available box types in list F . For box type r where $r \in I$ & $r \neq i$, the maximum box number k_r to be stacked onto this 1 box of type i satisfies such relationship

$$k_r \leq \frac{H - h_i}{h_r} < k_r + 1.$$

Therefore, the gap between the top of the stack and top of the consolidated empty block P_e can be defined as

$$H - h_i - h_r \cdot k_r.$$

In case that multiple dimensions can be vertically placed, all available orientations are tried for both box type i and r and the smallest gap of all orientation combinations is recorded. Same loop is applied for and ended till k_i boxes of type i . All such gaps are compared, and those box types generating the smallest gaps with the current box type i are recorded. We record up to three best matches for current box type i . We loop through all available box types I in such a way. Using the same procedure, we find out the best combination in widthwise. The evaluation phase is based on the observation that low utilisation is usually caused by either wasted space in height or width, or in both as the worst case.

Once the evaluation of best combination phase finishes, we keep packing the current block P_e . From the available box types I which are listed by default sequencing R , we pick the first box type as our main box type. We move all the box types which fit well with the main box type during evaluation phase, namely sub box types, directly after the main box type. We then call CM for the first time to pack this block and record the result. For the next rounds, we move down the position of the main box type each time and pack till its position is directly behind all its sub box types. The current main box type is back to its original sequencing position before the next main box type is selected. The next main box type is picked according to the default order, and current block is repeatedly packed under the same process. We limit that only the first half of the original box types in list F can be set as the main box type. We compare all the results, and pick the one with the highest utilisation.

Here we split original box types to explore more solution space within a neighbourhood. For the result with the highest utilisation, we split each box type equally into 2 batches (i.e. 2 new box types with exact same box dimensions) with same box numbers. For example, we split box type A into box type A_1 and A_2 . We then place each batch using the current order before calling CM. For example, a list of box type $ABCD$ is split into $A_1B_1C_1D_1A_2B_2C_2D_2$. We also try to split them into 3 batches and 4 batches. In case that box numbers can not be divided equally, there will be a batch with those extra boxes which can not be divided. We

however do not split awkward box types. This is because awkward box types tend to result in bad packing and we do not want them to be mixed up with other box types which can potentially have better packing.

We compare these three results with the original non-split best result, and note the one with highest utilisation. The new result is compared with U_H , which is generated in Step 36 of Algorithm 1a. We only accept the new result if it is higher than U_H or an improvement is already observed during the packing of a previous block, otherwise the new result is rejected and the current block is rotated to the end of the consolidated empty block list.

5.2.6 Kick Phase

Two options of kick, namely small kick and big kick, are provided in this phase. If searching within the neighbourhood does not result in an improvement, rather than have a big kick immediately we design a so-called small kick in the hope of expanding current neighbourhood further before kick into a completely different neighbourhood. In the case of failing to reach the criteria of small kick, big kick is carried out.

For small kick, we randomly loop through current J packed blocks and unpack those ones with utilisation U_j lower than the current U_m . The current U_m comes from last running of Phase 3 in Algorithm 1a. Again an upper bound of 25% and a lower bound of 10% of total box volume are put on the unpacked box volume to make sure we do not unpack too many or too few boxes. If the next unpack will result in the exceeding of the upper bound, the selected block will be dropped and another block will be randomly selected without violating the allowed volume range of [10%, 25%]. The unpacking process terminates once the current unpacked volume exceeds the lower bound of 10%. In the case of failing to reach the lower bound, the small kick process is terminated and we start implementing big kick.

The big kick phase is designed to give those well-packed blocks same chance to be unpacked. This is because we believe that the best packed blocks (tightly packed with a extremely high utilisation compared to others) can be considered as the cause of those poorly packed blocks, as it takes away the opportunities that other blocks will have a moderate utilisation at the cost of a lower utilisation of the best packed block. Although not in all cases, the best packed blocks are usually the first packed ones. Regardless of the priority order of different box types, considerably good combinations can always be found when all box types are available to be chosen from. Thus, it results that the later packed blocks only have

a few box types to be chosen from. With such few box types, there may be cases that all possible combinations are poor enough to leave large gap waste in the height or width.

During the big kick, we randomly select blocks to unpack from the current packed. Only a lower bound of 10% of total box volume is put on the unpacked box volume. Once the lower bound is reached, the big kick process ends. The reason for setting 10% is to make sure that enough of the solution is destroyed. We also set up a list to record all the blocks unpacked during last two kicks so that we do not repeatedly kick into the same neighbourhood. We record the length and the container number of those blocks to prevent situations where blocks with same length and container number happen to be selected for unpacking during current kick. Thus, same blocks, whether with same or different boxes packed inside, are not to be unpacked during every kick.

5.2.7 Generate Final Solution

Upon reaching the running time limit, we start to generate the final solution by packing the remaining boxes in list F . The final solution is generated based on the best partial solution recorded so far. We firstly loop through all existing E containers to check if there is still empty blocks available for remaining boxes. Once no such block is found, we then pack the remaining boxes into new containers. New containers are added one at a time based on the need.

5.3 Iterated Local Search with Descent and Randomness (ILSDR)

5.3.1 Notation

While many steps are the same as those in Iterated Local Search with Intelligent Neighbourhood, two new notations are introduced to reflect the new elements of Iterated Local Search with Descent and Randomness (ILSDR).

- F_{Split} be the new unpacked box list after the original box types in F are split
- O be the stopping criteria for the stage of Search within the Neighbourhood

5.3.2 Design and Framework

While keeping the framework structure similar to the ILSIN, we apply modifications with certain aims in mind. Firstly, we want to explore the impact of a more simple and direct acceptance criteria. Secondly, we try to find out if the intelligent neighbourhood designed in ILSIN does help improve results by implementing more randomness instead of manual effort when searching within a neighbourhood. Moreover, we question if it is worth accepting worse solution so in ILSDR we apply a descent move which only accepts improving results.

The general steps of this framework are presented in Algorithm 2. The algorithm can be broke down into five stages. First stage is the Generate Partial Initial Solution stage which is step 1. Stage of Form a New Neighbourhood expands from step 3 to 7. Step 8 presents the stage of Search within the Neighbourhood. The last two stages are the Descent Move and the Generation of Final Solution.

Algorithm 2 Iterated Local Search with Descent and Randomness (ILSDR)

- 1: Generate partial initial solution by using a fixed sequencing rule R or by using pre-assignment method
 - 2: **while** time limit is not reached **do**
 - 3: Identify Independent Blocks
 - 4: Update Best Solution
 - 5: **while** improvement is made **do**
 - 6: Randomly choose a block (or more if needed) to unpack
 - 7: Consolidate emptied blocks within their container
 - 8: Repeat the process of randomly splitting batches and packing until stopping criteria O is reached
 - 9: **end while**
 - 10: Apply descent move
 - 11: **end while**
 - 12: Generate final solution by packing the remaining boxes
-

5.3.3 Generate Partial Initial Solution

The partial initial solution is generated using the same method from ILSIN. It can be generated by either pre-assignment method or by default-priority method. Both methods are already described in detail in section 5.2.3.

5.3.4 Form a New Neighbourhood

Step 3 and 4 are the same as those in the ILSIN, and can be referred back to Phase 1 and 2 of Algorithm 1a. For step 6, rather than unpacking blocks using a set criteria we randomly choose a block to unpack. We choose only 1 block so the size of split batch is controllable. The unpacked boxes are added to the unpacked box list F . However, if less than 3 box types are remaining unpacked, we randomly choose another block till no less than 3 box types remain unpacked. The reason for this is because there may not be a difference for only 2 box types even after we split batches. We then consolidate blocks within the same container, as well as those empty blocks within the same container in step 7.

5.3.5 Search within the Neighbourhood

From the unpacked box list F , we randomly pick a box type and a random number of boxes of that type. A new box type is created this way, and the chosen box number is deduced from F . This step is repeated till all boxes in the list F are transferred to the new packing list F_{Split} . We then repack the empty block with the new list F_{Split} . In the case of multiple empty blocks, we randomly pick a block each time till all empty blocks are packed. The described process is repeated until stopping criteria O reached.

We form our stopping criteria O as follows. If the number of unpacked box types is no more than 30% of the number of original box types, we give t_1 seconds running time for the whole repeating process. If the number of unpacked box types is more than 30% but not exceeding 60% of the number of original box types, we give t_2 seconds running time. If the number of unpacked box types is more than 60% of the number of original box types, we give t_3 seconds running time. If the number of unpacked boxes is more than 40% of the total number of original boxes, we double the stopping time decided based on the percentage of remaining box types. t_1 , t_2 and t_3 are user-defined parameters, and the exact value we use can be found in section 5.6.1. We form stopping criteria in such a way so unpacked boxes with more box number and/or more box types are allocated with more time, i.e. $t_1 < t_2 < t_3$.

5.3.6 Descent Move

Once the empty block(s) is repacked, we go to Step 3. If improvement is found, we note the last packed blocks to avoid them being unpacked immediately in the next round. Then we start from Step 6 to randomly choose a block to unpack. If improvement is not found,

all blocks recently packed in Step 8 are unpacked. One of them is randomly selected and restored to its former packing. If the move had only packed 1 block, this block is restored. We then start from Step 6. This design is to have unpacked blocks as minimum as possible so the size of split batch is controllable.

5.3.7 Generate Final Solution

Upon reaching running time limit, we firstly loop through all existing containers to check if there is still empty blocks available for remaining boxes. We then pack the remaining boxes into the new containers. New containers are added one at a time based on the need. We use the same method in Step 8 treating the whole new container as a block. Once all the remaining boxes are packed, we loop the whole process again starting another round of splitting batch until the stopping criteria reached. Same stopping criteria based on the number of remaining box types and boxes in Step 8 apply.

5.4 Iterated Local Search with Simple Neighbourhood (ILSSN)

5.4.1 Notation

While Iterated Local Search with Simple Neighbourhood (ILSSN) is a local search algorithm just as Iterated Local Search with Intelligent Neighbourhood (ILSIN) and Iterated Local Search with Descent and Randomness (ILSDR), the design of ILSSN has no relation with those two. Therefore, we introduce a set of new notations for our ILSSN algorithm.

- R be a fixed sequencing rule
- S_i be the initial solution
- S_c be the current solution
- S_b be the best solution
- C be the set of packed containers
- c be the packed container c , where $c \in C$
- B be the set of box types

- b be the box type b , where $b \in B$
- c_t be the target container
- b_t be all the chosen boxes for a target box type within the target container c_t
- n be the number of chosen target boxes b_t
- L be the container length used in the target container c_t
- L' be the new container length used in the target container c_t
- c_u be the untarget container, where $u \in C$ & $u \neq c$
- b_u be all the boxes within the untarget container c_u
- b'_u be the new set of all the boxes within the untarget container c_u
- U be the utilisation in the untarget container c_u
- U' be the new utilisation in the untarget container c_u
- r be the number of chosen target boxes b_t during kick - n Random Move
- b''_u be the new set of all the boxes within the new untarget container c_u during kick - fill Target Container Move
- b''_t be the new set of all the boxes within the new target container c_t during kick - fill Target Container Move

5.4.2 Design and Framework

The idea behind ILSSN is to design a straightforward framework contrary to ILSIN. In ILSIN, unlimited neighbor pool makes it impossible to search extensively within the same neighborhoods. That means the local optima may be missed and even if you are in a good neighbourhood the chance to reach the local optima is low. We therefore reduce the neighbourhood to a manageable size and search within it extensively. Also, we do not implement the idea of dividing container into separate sections as we realize that by only repacking a section of container the size of the section itself may be a limitation to achieve the desired result. The design of the framework is based on the idea of emptying the least full container c_t . The earliest paper we can find using this strategy that reduces the number of containers one at a time is Bengtsson (1982).

The general steps of this framework are presented in Algorithm 3. The algorithm can be divided into four main components. Firstly, a complete initial solution is generated. Inter Container Optimisation is then applied to search within the neighbourhood, i.e. from step 3 to 32. Two alternative kicks are proposed to explore other neighbourhoods. Intra Container Optimisation is optional and can be applied before kick or after reaching stopping criteria.

5.4.3 Generate Complete Initial Solution

Different from ILS with Intelligent Neighborhood which generates a partial initial solution, we general a complete initial solution S_i which packs all boxes into containers. We use default priority R to place box types into a fixed order, and repeatedly call CM to pack boxes until all boxes are packed. The initial solution S_i is automatically recorded as the best solution S_b , i.e. $S_b = S_i$.

5.4.4 Inter Container Optimisation

We set the least full container with minimum utilisation rate as the target container c_t , and unpack it. We then choose n boxes of target box type b_t by its pre-decided priority, the number n is decided in such a way that we wish to move a batch of boxes rather than 1 box each time. We therefore divide container height (H) by the height of target box type b_t . In the case where more than one box dimension can be vertical, the largest dimension is always picked. A figure will be obtained indicating the number of boxes in a column. If the available box number is no less than this figure, pick the figure. Otherwise all the available boxes are picked.

Once we have the target container c_t and n target box type b_t selected, we then select an untarget container c_u by indexical order from all available containers along with all its untarget boxes b_u . We insert b_t in each position within the sequencing of b_u it can be while keeping the sequencing among b_u unchanged. We pack $b_u + b_t$ and accept the first improvement. For example, we assume box types A, B and C are in the untarget container and target box type is T. We have four positions to insert T, i.e. TABC, ATBC, ABTC, and ABCT.

It is worth mentioning that our coding is specifically designed to eliminate unnecessary runs which will not generate better results. When packing n boxes of target box type b_t into a untarget container which initially doesn't contain the same target box type b_t , the current

Algorithm 3 Iterated Local Search with Simple Neighborhood (ILSSN)

```

1: Generate initial solution  $S_i$  under fixed sequencing rule  $R$ , and set  $S_b = S_i$ 
2: while time limit is not reached do
3:   for all container  $c \in C$  do while  $C > 1$  do
4:     Pick target container  $c_t$  with minimum utilisation (and unpack it)
5:     for each box type  $b \in B$  within  $c_t$  do while  $B \neq \emptyset$  do
6:       Choose  $n$  boxes of target box type  $b_t$  by its pre-decided priority, the number
        $n$  is decided under certain Rule
7:       for each remaining  $c \in C - c_t$  do
8:         Select an untarget container  $c_u$  (and its all boxes  $b_u$ ) by index order
9:         for each priority  $b_t$  can be (while keeping relative priority among  $b_u$ ) do
10:          pack  $b_u + b_t$  and accept the First Improvement
11:          if all boxes can be packed ( $b'_u = b_u + b_t$ ) then
12:            Update solution and return to Step 6 providing the target con-
            tainer hasn't been emptied
13:          else
14:            Check utilisation rate in the untarget container  $c_u$ 
15:            if  $U \leq U'$  then
16:              Repack target container  $c_t$  with remaining boxes  $b_u + b_t - b'_u$ 
17:              if  $b_u + b_t - b'_u$  are all packed in  $c_t$  then
18:                check length used in the target container
19:                if  $L > L'$  OR  $L = L'$  and  $U' < U$  do
20:                  Update solution and return to Step 6 providing the
                  target container hasn't been emptied
21:                end if
22:              end if
23:            end if
24:          end if
25:        end for
26:      end for
27:    end for
28:  end for
29:  Compare  $S_c$  with  $S_b$ 
30:  if  $S_c < S_b$  then
31:     $S_b = S_c$ 
32:  end if
33:  Apply Optional Intra Container Moves
34:  Apply KICK (i.e. nRM or fTCM) to jump out of current local optimal
35: end while
36: Apply Optional Intra Container Moves

```

untarget container is temporarily abandoned and the packing moves on to the next untarget container if the same exact n boxes of target box type b_t is return as unpacked and the untarget container remains the same utilisation. However, if the current untarget container initially already contains the same target box type b_t , the packing will carry on as these n boxes of target box type b_t might be all or partially packed later.

If all boxes can be packed, i.e. $b'_u = b_u + b_t$, we update solution and go to step 6 in Algorithm 3 to select n boxes of a new target box type b_t providing the target container hasn't been emptied. However if there are boxes cannot be packed, check utilisation rate in the untarget container. If utilisation keeps the same or improves, i.e. $U \leq U'$, we repack target container c_t with remaining boxes $b_u + b_t - b'_u$. We then check the length used in the target container. If the length used in target container is improved, i.e. $L > L'$, or if the length used in target container remains the same but utilisation in untarget container improves, i.e. $L = L'$ and $U' < U$, we update the solution and go to step 6 in Algorithm 3. If no improvement can be found in this untarget container, we go to step 8 in Algorithm 3 moving on to the next untarget container by indexical order. If no improvement is found in all untarget containers, we mark the current target box type b_t unstackable and go to step 6 in Algorithm 3.

In the case of the target container c_t is emptied, we go to step 4 in Algorithm 3 picking a new target container. When no further improvement can be made during Inter Container Optimisation phase, we start the stage of Intra Container Optimisation, which will be explained in the section 5.4.5.

5.4.5 Intra Container Optimisation

The aim for this stage is to repack the boxes already in the untarget containers and reduce each container's length used. Each untarget container is selected by indexical order. For each box type within the untarget container c_u , we lower its priority rank one position at a time. For example, we assume there are four box types A, B, C and D in the untarget container c_u . We firstly pick box type A and lower its position one each time. We have three combinations, i.e. BACD, BCAD, and BCDA. We repack the untarget container c_u for each such position. If the length used is improved, i.e. $L' < L$, we update solution. The process is stopped if current box type reaches the bottom of the list. We then restore the list to the initial list and pick the next box type repeating the same steps. Taking the previous example, we now move down box type B's position one each time, i.e. ACBD, and ACDB. We accept the best improvement after trying out all the cases.

If any improvement is spotted during the Intra Container Optimisation stage, the Inter Container Optimisation stage is re-started from step 6 in Algorithm 3 as target container c_t is already selected. A kick is implemented when the re-started Inter Container Optimisation stage ends. If no improvement is found during the Intra Container Optimisation stage, we implement a kick, either n Random Move or fill Target Container Move, to jump out of current local optimal.

Intra Container Optimisation is optionally implemented before kick or after the run time limit is reached. Details of this option can be found in section 5.6.1.

5.4.6 Kick

We implement two alternative kick moves. For the first called n Random Move, we randomly select a random number r of box type b_t within the target container c_t and randomly select an untarget container c_u . We pack $b_t + b_u$ in the untarget container c_u and always list b_t ahead of b_u to ensure they are packed. The new packing in c_u is described as b'_u . We then repack target container c_t with the remaining boxes $b_u + b_t - b'_u$ and update solution. While C is defined as the total number of packed containers for Algorithm 3, $C - 1$ is the number of untarget containers. We implement such random move for $C - 1$ times to ensure the kick is significant enough to jump to another neighbourhood.

For fill Target Container Move, we pick a random untarget container c_u . We then swap target container c_t and the selected untarget container c_u . The selected untarget container c_u thus becomes the new target container c_t , while the target container c_t becomes untarget container c_u . We pack all boxes $b_t + b_u$ from the target container c_t and the untarget container c_u into the untarget container c_u . If all boxes $b_t + b_u$ can be packed, we reduce the number of containers by 1. If not all boxes can be packed, we pack the remaining boxes $b_t + b_u - b''_u$ into target container c_t . If still not all boxes $b_t + b_u - b''_u$ can be packed, we add in an extra container and pack the remaining boxes $b_t + b_u - b''_u - b'_t$ into it.

5.4.7 Obtain Final Solution

Unlike ILSIN and ILSDR, ILSSN does not require extra time to generate the final solution as the algorithm generates full solutions every time rather than partial solution. Therefore, the best solution when the run time limit is reached automatically becomes the final solution.

5.5 The Proposed Beam Search Algorithm

In the previous sections, we designed those iterated local search algorithms of varying complexity. Also, local search serves as an improvement heuristic which corrects the mistakes made in the initial solution. As reviewed in chapter 3.11, the beam search proposed here works in a different way to all the previous algorithms. It aims to exploit what CM is originally designed for and good at, i.e. packing a single container. Beam search works on a few partial solutions in parallel and builds from there by gradually adding those new elements which help retain the quality of the partial solutions. The quality of each new element is based on container utilisation and its impact upon the overall utilisation, which is the criterion when judging the quality of the final solution for Homogeneous Container Problem. Each partial solution is built into final solution through adding a new container at each stage. A few complete solutions are only generated at the end of the algorithms, and the best one is selected as the final solution. While the general beam search practice adds a new element in each node, certain modification has to be made in our beam search implementation.

Other beam search implementations for cutting and packing problems include irregular strip packing problem (Bennell and Song, 2010). They consider a single piece with a specific orientation as a new element. All such options are presented as child nodes to each parent node. The number of levels in the tree graph equals to the number of pieces in that run (i.e. instance). The 3D container packing problem potentially involves a much larger number of boxes in a single instance and multiple large objects (containers). Also, the cost of computational time spent on each node by running Cargo Manager (CM) is much higher. These two facts force us to control the depth and width of the search tree very carefully. Therefore, our search tree represents the sequencing order of fully packed containers. Using figure 3.2 to illustrate our tree, each node in the search tree corresponds to a fully packed container except for the root node. The root node contains zero element, i.e. no container nor box. In this way, the tree's depth is controlled as the number of containers packed. A branch from a parent node (beam) towards a child node represents the decision to pack the next container with the boxes packed using a specific sequencing. Such decisions are made based on various strategies which we will discuss later in detail. As the number of child node branching from each parent node (beam) has a significant impact on the tree's width, it is not possible for us to list all the possible sequencing combination. Hence, we rely on the strategies to limit the excessive choices of such combinations.

Once all child nodes are generated at a level, both local and global evaluations are used to decide which nodes are the better ones to become beam for the next level. Given a filter

width α and beam width β , the local evaluation compares all child nodes from the same beam on their utilisation and length used. The higher the utilisation, the better the child node is. In the case of tie, the child node with shorter length is considered better. The best α child nodes from each beam are the filler nodes and selected for global evaluation. A further check is made to detect any repetitive node selected during local evaluation. A repetitive node is defined as a container of the same size packed with same number of boxes in the same sequence. Once such nodes are detected, they are not allowed to be a filter node so different nodes can be selected as filters. The global evaluation function continues to pack the remaining boxes for each filter node. The remaining boxes are placed within their original batch, and the box types are ordered using the fixed sequencing rule R which is also used in the previous local search algorithms to generate initial solutions. As figure 5.1 shows, global evaluation is carried on for filter node B. We pack the remaining boxes till all of them are packed. In this case, four containers are used in global evaluation for filter node B. The global evaluation is the total container length used in the final solution (L_t). It is calculated as the sum of the shortest length used across all containers and the full container length of the other containers. Assume container 4 has the shortest length used L_s , L_t in this case is the sum of L_B , L_1 , L_2 , L_3 and L_s where L_B , L_1 , L_2 and L_3 are identical. According to the global evaluation, the best β nodes across all the filter nodes are selected as the new beams for the next level. In the case when the number of total filtered nodes is less than the specific beam width, all these filtered nodes are used as beams for the next level.

Consider figure 3.2 as an example. Filter width α is 3, and beam width β is 2. Each parent node generates four child nodes. Four containers A, B, C and D are packed in parallel in level 1, where all boxes are available for packing. Container C is pruned after local evaluation, and after global evaluation B and D are selected as the beams for level 2. A total of eight containers are generated in parallel in level 2. Container a, b, c, and d are packed with the remaining boxes left after container B, and container e, f, g, and h are packed after container D. Container a and h are pruned after local evaluation, and after global evaluation container b and f are selected as the beams for level 3. As the result of such implementation, the nodes at the lowest level of the tree represent the last packed containers and the whole solution is found by tracking a path through the tree. The search tree stops at the level where it is observed that all boxes are packed in one or more nodes. For those nodes where all boxes are packed, one's parent nodes at all previous levels are backtracked. The branch with the shortest L_t (total container length used) is selected as the best solution.

As we mentioned above, we need to control the width and depth of the search tree. While depth is taken care of by making every child node a single packed container, the tree width is

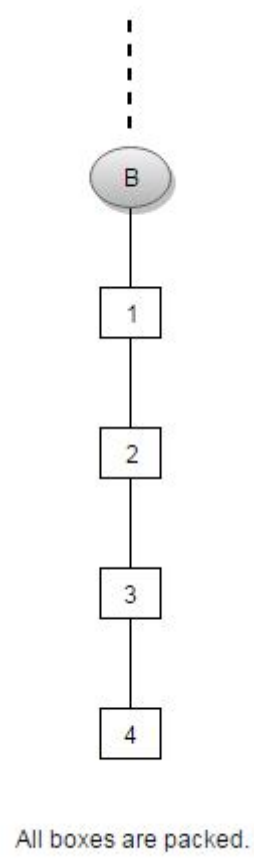


Fig. 5.1 Global Evaluation for Homogeneous Container Problem

based on the beam width and the number of child nodes generated from each beam. We will experiment and set up beam width as a parameter in computational results section 5.6.3. As for the number of child nodes, we design various strategies to help make decisions on generating better child nodes since it is impossible to list all the feasible sequencing combinations. Overall we propose seven strategies. All strategies are used for generating child nodes from each beam.

Strategy 0 packs a single container twice, and generates two child nodes. Under this strategy, all unpacked boxes remain in their original batch as defined in section 4.3.1. We set up two fixed sequencing combinations based on the outcome of our experiments on the impact of sequencing in section 4.5.2. For instances only allowing side rotation, box types are firstly sorted non-increasing by the single box volume of each box type. In the case of a tie, the box type with larger height is given higher priority. Base area serves as a third-tier tie in the case of same height. Box Type with larger base area has higher priority. For the second node, Box types are sorted non-increasing by the height of each box type. Single box volume and base area are taken as second-tier and third-tier tie accordingly.

Strategy 1 similarly places all boxes within their original batch. However, the sequence of box types is sorted randomly. The same process can be repeated multiple times to generate as many nodes as the user desires. This same rule applies to the rest five strategies. Multiple child nodes can be generated by repeating each one of the strategies.

Strategy 2 takes the concept of randomly sorting sequence of box types with single box type can be split into more box types. The number of boxes getting split into new box type is randomly selected each time. A maximum of 100 box types is allowed to make it a realistic strategy as too many box types only worsen the cost of computational time.

Strategy 3 is the modification of strategy 2. As a single box type in strategy 2 can be split into many new box types, we feel that the possibility of having so many box types with only a few boxes within each type may not necessarily lead us to better packing. We therefore decide to split each original box type into either 2 or 3 batches with random box number selected for each new type. All the new box types are sorted into a random sequence. Under this strategy, it still allows batch splitting which will lead us into different solution spaces. Also, it does not make the packing too heterogeneous. Fewer box types in larger batches will save computational time and may keep the good homogeneous block while exploring into new solution spaces.

Strategy 4 adopts most of strategy 3's practice, it gives larger box types higher priority in the sequencing compared to smaller box types. Here is how we divide boxes into these two

categories. All box types are sorted in non-increasing order of single box volume. Types in the first half of the list are categorised as larger box types, and the rest are smaller box types. We apply strategy 3 on those larger box types first to make sure they have priority over smaller box types. After all larger box types are re-arranged into new types, strategy 3 is then applied on the remaining smaller box types and the new types created are added after those previous larger types.

Strategy 5 adds the concept of column and row onto strategy 4. Whenever a random number of boxes are selected to form a new box type, it is reduced to the closest number which can form one or more columns or rows given the height and width of container. The idea behind this modification is that a homogeneous column or row is considered a nearly perfect partial solution, and by modifying the random number we hope to increase the chances for obtaining such partial solutions. The concept of splitting into only 2 or 3 batches is removed in this strategy so that more new types are allowed to hopefully create homogeneous columns or rows.

Strategy 6 embraces the idea of packing those we consider difficult boxes ahead of any other boxes. Once all filtered child nodes are selected after local evaluation, the global evaluation function continues to pack the remaining boxes left after each child node. The boxes packed in the last container are considered as difficult boxes and are recorded within the box types as when they are packed. Beams are then selected among these filtered child nodes. At the next level, when child nodes are generated, strategy 6 is used alongside of other strategies by placing those difficult boxes at the top of the list. These difficult boxes are placed within their original batch and are randomly sorted using strategy 1. The remaining boxes are re-arranged in the same manner. Different from other strategies, strategy 6 only applies starting from level 2.

Our beam search algorithm is summarised by the following procedure.

- B be the total set of boxes
- b_r be the remaining boxes to be packed
- T be the original box types (batches)
- L_t be the total container length used
- c be the number of packed containers
- S_c be the partial solution containing c packed containers
- b_c be the boxes packed in the first c containers

- S_u be the partial solution for the current $(c + 1)th$ container
- b_u be the number of boxes packed in the current $(c + 1)th$ container
- s_n be the proposed strategy n where $n = 0, 1, \dots, 6$
- α be filter width
- β be beam width
- $GLOBAL(S_c + S_u, b_c + b_u, B)$ be the global evaluation function that returns the total length used of all packed containers, where the packing order of the first $(b_c + b_u)$ boxes in the partial solution $(S_c + S_u)$ have been determined by the search tree through the choices of s_n and the remaining $(B - b_c - b_u)$ unpacked boxes placed within their original batch T which are ordered using the fixed sequencing rule R
- $LOCAL(S_u, b_u)$ be the local evaluation function that returns the utilisation of current packed container with container length used as tier breaker

Step 0 Set $c = 0$, $S_0 = \emptyset$ and root node as the single beam node. Initialize $b_r = B$.

Step 1 Node generation. Generate S_u (i.e. level $(c + 1)$ nodes) by adopting s_n where $n = 0, 1, \dots, 6$.

Step 2 Termination test. If one or more of the nodes has $b_u = b_r$, go to Step 6.

Step 3 Filtering. For each node compute $LOCAL(S_u, b_u)$ and select the best α nodes. Prune remaining nodes. If the number of available nodes is less than α , all nodes are selected as filtered nodes.

Step 4 Select beam nodes. Compute $GLOBAL(S_c + S_u, b_c + b_u, B)$ for each of the $\alpha \times \beta$ filtered nodes ($\beta = 1$ when $c = 0$). Select the best β nodes as beam nodes. If the number of filtered nodes is less than β , all filtered nodes are selected as beam nodes.

Step 5 Update sets. For each selected beam node, form S_{c+1} by adding the beam node S_u to S_c , and remove b_u from b_r . Let $c = c + 1$, and go back to Step 1.

Step 6 Final solution. Among all the nodes where $b_u = b_r$, select the one (along with the whole branch) with the shortest total container length used L_t .

5.6 Computational Results

In this section, under the chosen sequencing and heuristic derived from section 4.5.2 and 4.5.3 solutions are generated by our three iterated local search algorithms and beam search algorithm for homogeneous containers problem. Firstly, we discuss the experimental design for our iterated local search algorithms. Secondly, we generate and compare results under different conditions for all three local search algorithms. We then generate results under different parameter settings for beam search. Finally, we compare the best results from iterated local search and beam search with each other and against control-set results. The new benchmark data sets generated in chapter 4.4.2 are used to generate all the results. We coded all our algorithms in Java 1.8.0_05 and all the instances are run on an Intel(R) 2.60 GHz PC with a 4.00 GB RAM.

5.6.1 Iterated Local Search Experimental Design

For all three Iterated Local Search algorithms, sequencing combination VHN is selected as fixed sequencing rule R, because it is the combination which generates the best results through our experiment on the impact of sequencing and heuristic in section 4.5.2 and 4.5.3. Based on the results from the same experiment, we choose H2 as our constructive heuristic. The main reason for choosing only one heuristic is that we need to use the same constructive heuristic throughout the whole experiment to keep consistency.

For Iterated Local Search with Intelligent Neighbourhood (ILSIN), we decide the parameter E , which is used during generating the partial initial solution, based on the data sets we are running. As our benchmark data sets are generated to have their overall box volume match with the volume of either five containers or ten containers that all containers are 40ft Standard Steel Maersk container. Therefore, parameter E is 5 when data sets s1c5 and s2c5 are running while E is 10 when we are running data sets s1c10 and s2c10. We set parameter U_m as our experience and the experience of other researchers indicate that 80% utilisation is low in these problems. Therefore, starting to improve blocks with utilisation lower than 80% seems to be a good starting point. A running time limit of 500 seconds is set. The 500 seconds are set based on two reasons. The running time of 500 seconds is widely implemented in many literatures. Moreover, in practice a time span of 500 seconds seems to be a reasonable waiting time for clients to get a solution. Upon the time reached, extra time is given for algorithm to finalize the solution.

Iterated Local Search with Descent and Randomness (ILSDR) is run under the same setting

		default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
Picking Order	By Non-increasing Single Box Volume By Non-increasing Total Box Volume By Non-increasing Box Number	X	X	X	X	X	X	X	X	X	X
The Number of Boxes Selected	A Single Box A Single Column of Boxes A Wall of Boxes 11 Column of Boxes All Boxes	X	X	X	X			X	X	X	X
Kick	n Random Move (nRM) fill Target Container Move (fTCM)	X		X	X	X	X	X	X	X	X
Intra Container Optimisation (ICO)	No ICO Before Each Kick (i.e. At Local Optimal) At the End of 500 seconds Running Time	X	X	X	X	X	X	X	X	X	X

Table 5.1 Experimental Design for Iterated Local Search with Simple Neighbourhood

as ILS with Intelligent Neighborhood. As described in section 5.3.5, a running time of t_1 seconds is set when the number of unpacked box types is no more than 30% of the number of original box types. A running time of t_2 seconds is set for the number of unpacked box types which is more than 30% but not exceeding 60% of the number of original box types. When the number of unpacked box types is more than 60% of the number of original box types, a running time of t_3 seconds is implemented. We set the value of t_1 , t_2 and t_3 as 10, 20 and 30 seconds respectively so that unpacked boxes with larger number of box types get more running times.

We design our experiments to run Iterated Local Search with Simple Neighbourhood (ILSSN) under three conditions. The first condition is when selecting the target box type, the potential box type list can be ordered by non-increasing single box volume, by non-increasing total box volume, or by non-increasing box number. When deciding the number of target boxes, we can pick a single box, a single column of boxes, a wall of boxes, 11 column of boxes, or all boxes from the target box type. Due to the fact that only 1 box dimension can be vertical in all instances of data soton1, the number of boxes in a single column is fixed. Given the container size and the boxes sizes in data, the smallest box type can form 17 columns. Therefore, we choose 11 column as a middle ground to fill in the gap between a wall of boxes and all boxes from the target box type in the experimental design. The second condition concerns two options for the kick: n Random Move (nRM) and fill Target Container Move (fTCM). The last condition is Intra Container Optimisation (ICO) part. We run under no ICO at all, ICO before each kick, and ICO when 500 seconds running time is reached. The whole experimental design is summarized below as well as their notations in table 5.1. Each column indicates an experiment with their notation shown in the first row running under which conditions. For example, the default run uses the conditions of: non-increasing single box volume, a single column of boxes, n Random Move (nRM) and No ICO.

5.6.2 Iterated Local Search Results

In this section, we tested all three models on both dataset soton1 (s1) and soton2 (s2), i.e. 10 instances from boxtype5 (b5), boxtype10 (b10) and boxtype25 (b25) for both container5 (c5) and container10 (c10). Each instance is run 5 times, and the average value is taken. For ILS with Intelligent Neighborhood, all instances are run under two different initial solution generation method, i.e. default sequencing (default) and pre-assignment (pre-assign). Because only partial initial solution is generated under ILSIN (and ILSDR), we only compare two final solutions from each other at this stage. A complete initial solution however is generated under algorithm ILSSN. Therefore, a final comparison among final solutions from all three algorithms and this initial solution is drawn in Table 5.10 and Table 5.11. All results are reported in terms of percentage utilisation. As we explained earlier in the algorithm section that the utilisation is defined as the utilisation of total packing length. The total packing length is calculated by the shortest container length used among all packed containers plus the sum of the full length of the rest packed containers.

Table 5.2 shows the results of ILSIN for soton1. Results generated by default sequencing are better in general. For data s1c5b5, s1c5b10, s1c5b25, s1c10b5, s1c10b10 and s1c10b25, results generated by default sequencing are 0.84%, 1.41%, 3.36%, 1.18%, 1.39% and 1.82% better than those generated using pre-assignment method respectively. Time spent on evaluating neighbourhood is at most 0.1% of overall running time. This shows that the time spent on neighbourhood evaluation is far from being a constraint on the overall running time. We monitor such measure because we try to avoid the situation where the evaluation goes on a huge amount of time under the condition of not knowing how exactly CM's constructive heuristic works and as a result of such situation that the real and only precise evaluation, which is running the standalone blackbox, does not take place enough than it should be.

We define split batch improvement rate as the rate of improved split batch result during the full running time. As described in section 5.2.5, the results from split batch are firstly compared with the non-split best result and the best one is then compare with U_H from section 5.2.4. Such comparison counts as 1 try, and a split result better than U_H is noted as an improvement. For default results, the split batch improvement rate is 8.14%, 9.19%, 8.71%, 3.56%, 9.01% and 8.05% for data s1c5b5, s1c5b10, s1c5b25, s1c10b5, s1c10b10 and s1c10b25 respectively. This shows that the strategy of split batch has a positive impact on the result.

The number of times a kick is performed is 29.32 (with a standard deviation of 32.07), 8.32

(9.91), 1.92 (1.83), 14.42 (12.20), 4.94 (6.96) and 0.76 (1.02) for data s1c5b5, s1c5b10, s1c5b25, s1c10b5, s1c10b10 and s1c10b25 respectively. Instances with fewer box types have considerable more number of kicks than those with more box types. Also, instances requiring more containers have a lower number of kicks. The standard deviations, however, suggest that the number of times a kick is performed is extremely instance-specific under this algorithm. Instances within the same data cluster, e.g. s1c5b10, have totally different numbers of kicks among them. In short, generating partial initial solution by default sequencing leads to better final results in general when comparing to that generated by pre-assign strategy.

Instance	s1c5b5		s1c5b10		s1c5b25		s1c10b5		s1c10b10		s1c10b25	
	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign
1	87.299%	86.706%	81.064%	80.535%	81.398%	79.599%	82.715%	81.992%	80.362%	79.510%	81.462%	80.866%
2	72.421%	72.289%	89.277%	86.964%	83.742%	81.284%	87.011%	87.092%	81.978%	80.612%	89.162%	84.826%
3	91.872%	87.825%	71.545%	71.489%	86.984%	84.562%	80.149%	78.666%	77.398%	76.043%	80.322%	79.530%
4	90.747%	89.310%	74.826%	74.951%	84.454%	83.694%	72.247%	70.827%	76.876%	75.929%	82.862%	80.533%
5	91.972%	91.245%	80.176%	79.475%	82.428%	77.617%	76.995%	76.995%	85.396%	84.724%	79.638%	78.278%
6	88.108%	87.052%	89.586%	89.128%	84.751%	81.605%	84.083%	83.846%	89.569%	88.780%	82.664%	83.701%
7	81.599%	81.486%	86.342%	84.867%	81.729%	80.486%	88.029%	85.579%	84.362%	84.347%	81.873%	82.638%
8	75.281%	77.684%	84.167%	82.064%	86.287%	83.387%	75.137%	73.041%	74.019%	73.930%	76.824%	75.419%
9	80.646%	80.318%	91.671%	89.438%	91.644%	86.952%	75.799%	75.584%	79.927%	77.791%	89.693%	87.021%
10	78.297%	77.353%	93.648%	91.674%	87.315%	83.882%	76.392%	75.630%	84.619%	81.712%	84.339%	81.218%
Average	83.824%	83.127%	84.230%	83.059%	85.073%	82.307%	79.856%	78.925%	81.450%	80.338%	82.884%	81.403%

Table 5.2 soton1 Results for ILS with Intelligent Neighborhood

Table 5.3 shows the results of ILSIN for soton2. The similar outcome is observed compared to that for soton1, as results generated by default sequencing are better in general. Only that the number of kicks suggests that the search is barely extended outside of the first neighbourhood. The number of kicks is 1.50 (with a standard deviation of 2.46), 4.04 (11.27), 0.12 (0.48), 0.18 (0.56), 0.46 (1.57) and 1.32 (3.89) for data s2c5b5, s2c5b10, s2c5b25, s2c10b5, s2c10b10 and s2c10b25 respectively. We however restrained ourselves from setting a longer running time so that the search could extend to more neighbourhood. We intend to show that within the same running time, the same algorithm may yield inconsistent performances towards different data sets. As we mentioned in section 4.4.2, data set soton1 represents electronic products which are much larger than products under ‘Paper, Computer and Office Supplies’ category represented by data set soton2. To sum up, generating partial initial solution by default sequencing leads to better final results in all instances compared to that generated by pre-assign strategy.

All results of ILSDR are reported in terms of percentage utilisation. Table 5.4 shows the results of ILSDR for soton1. Results generated by default sequencing are better in general. For data s1c5b5, s1c5b10, s1c5b25, s1c10b5, s1c10b10 and s1c10b25, results generated by default sequencing are 3.07%, 4.40%, 4.91%, 3.24%, 3.88% and 4.33% better than those

Instance	s2c5b5		s2c5b10		s2c5b25		s2c10b5		s2c10b10		s2c10b25	
	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign
1	93.237%	92.906%	92.475%	90.275%	90.692%	90.581%	95.506%	93.360%	94.269%	90.838%	94.094%	88.880%
2	92.607%	90.638%	94.720%	91.663%	94.384%	91.243%	93.664%	92.392%	93.691%	90.590%	93.537%	88.620%
3	90.450%	89.979%	80.460%	80.089%	94.345%	90.853%	94.259%	91.948%	95.972%	93.658%	93.148%	88.395%
4	90.599%	90.737%	93.954%	91.401%	95.766%	91.928%	95.087%	92.989%	94.810%	91.416%	93.027%	88.532%
5	91.434%	90.785%	91.450%	90.261%	94.128%	92.438%	91.608%	90.596%	95.733%	93.989%	92.628%	89.823%
6	94.028%	93.057%	94.455%	91.227%	94.457%	91.699%	93.809%	92.909%	94.810%	93.012%	94.084%	89.846%
7	95.310%	94.713%	92.627%	91.052%	93.125%	91.387%	90.586%	89.159%	94.117%	91.843%	80.973%	78.031%
8	93.333%	92.911%	93.727%	91.257%	90.722%	89.512%	95.485%	90.374%	91.960%	91.385%	92.350%	89.405%
9	88.586%	87.941%	95.629%	93.615%	94.140%	92.636%	91.634%	90.783%	95.157%	93.252%	94.054%	88.105%
10	93.174%	92.188%	93.541%	91.490%	92.467%	90.642%	93.618%	93.520%	96.460%	90.897%	93.153%	88.537%
Average	92.276%	91.585%	92.304%	90.233%	93.423%	91.292%	93.526%	91.803%	94.698%	92.088%	92.105%	87.817%

Table 5.3 soton2 Results for ILS with Intelligent Neighborhood

generated using pre-assignment method respectively. Time spent on splitting batch (excluding Cargo Manager running time) is at most 0.1% of overall running time. This shows that the strategy we design does not consume much running time. We define split batch improvement rate as the rate of improved split batch results during the full running time. Each split batch, no matter if it splits the same box type list or not, counts 1. For default results, the split batch improvement rate is 0.53%, 0.84%, 0.47%, 1.77%, 0.80% and 1.39% for data s1c5b5, s1c5b10, s1c5b25, s1c10b5, s1c10b10 and s1c10b25 respectively. This result shows that the design of split batch here is mediocre. The number of descents is 24.16 (with a standard deviation of 6.94), 27.44 (7.96), 30.50 (5.30), 30.84 (8.46), 32.80 (7.41) and 41.92 (5.60) for data s1c5b5, s1c5b10, s1c5b25, s1c10b5, s1c10b10 and s1c10b25 respectively. The number of descent increases with the increase of box type and container number.

Instance	s1c5b5		s1c5b10		s1c5b25		s1c10b5		s1c10b10		s1c10b25	
	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign
1	87.403%	86.184%	80.275%	77.868%	81.027%	77.798%	82.690%	81.727%	80.166%	78.315%	80.967%	77.894%
2	72.677%	71.819%	88.346%	84.202%	83.682%	79.435%	86.830%	83.735%	81.977%	78.704%	89.162%	80.778%
3	91.872%	85.834%	70.915%	71.055%	86.802%	81.531%	77.943%	76.456%	77.946%	71.779%	78.692%	77.064%
4	90.186%	87.191%	74.658%	73.839%	84.048%	82.376%	72.572%	68.266%	76.801%	74.702%	81.436%	79.172%
5	92.068%	90.428%	79.944%	78.295%	81.103%	76.581%	76.621%	76.743%	85.282%	83.268%	78.802%	75.617%
6	88.392%	83.657%	86.270%	81.192%	84.309%	81.248%	84.087%	76.106%	88.810%	85.601%	81.243%	80.685%
7	81.634%	80.179%	86.216%	81.428%	81.028%	79.437%	88.285%	84.009%	85.473%	82.338%	81.746%	80.988%
8	80.689%	77.496%	83.548%	80.071%	85.599%	81.807%	73.998%	73.442%	73.043%	72.707%	76.823%	73.813%
9	80.348%	79.710%	91.511%	85.860%	91.159%	85.411%	76.467%	75.591%	79.424%	76.163%	89.202%	83.466%
10	78.167%	75.788%	93.648%	86.345%	87.200%	80.728%	76.798%	75.252%	84.606%	79.605%	84.377%	78.854%
Average	84.344%	81.829%	83.533%	80.016%	84.596%	80.635%	79.629%	77.133%	81.353%	78.318%	82.245%	78.833%

Table 5.4 soton1 Results for ILS with Descent and Randomness

Table 5.5 shows the results of ILSDR for soton2. Results generated by default sequencing again are better in general. Similar to the outcome for soton1, randomly batch splitting is not an efficient design under the framework of ILSDR. Only a tiny portion of running time actually has a positive impact on the overall performance for soton1 and soton2. The number of descents is 20.72 (with a standard deviation of 3.89), 21.66 (4.87), 20.34 (4.25), 19.78 (2.79), 22.50 (4.09) and 23.00 (6.32) for data s2c5b5, s2c5b10, s2c5b25, s2c10b5, s2c10b10

and s2c10b25 respectively. Different from those for soton1, the number of descent remains rather the same regardless of the increase of box type and container number. The standard deviation indicates that the mean is a reasonable representer for all instances in the same data cluster for both soton1 and soton2. In short, generating partial initial solution with default sequencing leads to better final solutions for both data set soton1 and soton2.

Instance	s2c5b5		s2c5b10		s2c5b25		s2c10b5		s2c10b10		s2c10b25	
	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign	default	pre-assign
1	93.580%	93.253%	92.434%	91.615%	90.605%	89.648%	95.506%	91.521%	94.269%	90.476%	94.094%	89.206%
2	92.607%	91.714%	94.769%	92.709%	94.383%	90.230%	93.667%	89.866%	93.691%	89.850%	93.537%	89.313%
3	90.378%	89.763%	80.460%	80.184%	94.345%	90.192%	94.318%	90.573%	95.888%	89.599%	93.148%	89.330%
4	90.881%	91.029%	93.954%	92.459%	95.766%	91.229%	94.831%	89.219%	94.810%	90.790%	93.015%	89.079%
5	91.493%	91.448%	91.939%	90.844%	94.128%	91.542%	92.175%	91.735%	95.732%	90.334%	92.628%	88.687%
6	94.028%	92.442%	94.500%	92.233%	94.438%	91.580%	93.878%	90.110%	94.799%	91.042%	94.055%	89.299%
7	95.342%	94.516%	92.636%	90.786%	92.914%	88.941%	90.857%	90.125%	94.117%	90.475%	80.394%	79.070%
8	93.391%	92.404%	93.727%	92.193%	90.798%	88.651%	95.522%	91.873%	92.023%	89.767%	92.185%	89.335%
9	88.586%	88.011%	95.629%	93.968%	94.140%	91.585%	91.578%	90.708%	94.982%	90.834%	94.054%	89.947%
10	93.268%	92.537%	93.541%	91.884%	92.480%	89.745%	94.121%	91.871%	96.466%	90.215%	93.153%	88.332%
Average	92.355%	91.712%	92.359%	90.887%	93.400%	90.334%	93.645%	90.760%	94.678%	90.338%	92.026%	88.160%

Table 5.5 soton2 Results for ILS with Descent and Randomness

The results of ILSSN for data set soton1 are shown in Table 5.6 and Table 5.7. Algorithm generates the best results under experiment allBoxes apart from data s1c5b25 which has its best result generated under experiment 11Column. However, the result under experiment 11Column is just 0.009% better than the one under experiment allBoxes for data s1c5b25. For results run under allBoxes experiment, the number of CM executed is 3842, 3638, 2793, 4005, 3911 and 3865 for data s1c5b5, s1c5b10, s1c5b25, s1c10b5, s1c10b10 and s1c10b25 respectively. We record this in addition to support our findings in section 4.5.1. The number of kicks is 39.72 (with a standard deviation of 13.76), 12.76 (2.30), 2.94 (2.02), 24.58 (9.94), 8.66 (2.22) and 2.86 (0.93) for data s1c5b5, s1c5b10, s1c5b25, s1c10b5, s1c10b10 and s1c10b25 respectively.

Similar to ILSIN, significantly fewer kicks are observed for instances having more box types and requiring more containers. Standard deviations suggest that instances in the same data cluster have similar number of kicks. In terms of results, one of the observed reasons for good performance when selecting all boxes is that once all boxes from target box type b_t are selected, any number of boxes might be packed into untarget container c_u because those particular number of boxes might form a good enough block. When fewer, e.g. a column of, boxes is selected, they might not be packed because the column is not considered good enough by CM algorithm due to issues like stability.

The results of ILSSN for data set soton2 are shown in Table 5.8 and Table 5.9. Algorithm generates the best results under experiment allBoxes for data set s2c5b5, s2c5b10, s2c10b5 and s2c10b10. Data set s2c5b25 has its best result generated under experiment 11Column.

s1c5b5	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	87.793%	87.945%	87.453%	87.572%	87.967%	87.847%	87.751%	87.676%	87.593%	87.685%
2	73.085%	72.677%	73.138%	72.968%	72.685%	73.090%	73.028%	72.938%	72.975%	73.018%
3	91.872%	92.233%	91.872%	91.872%	92.353%	92.012%	91.895%	91.872%	91.872%	91.872%
4	90.326%	90.873%	90.272%	90.521%	91.566%	91.240%	90.932%	90.166%	90.336%	90.428%
5	92.541%	91.972%	92.521%	92.410%	92.234%	92.755%	92.839%	92.284%	92.554%	92.504%
6	88.284%	88.242%	88.258%	88.165%	88.643%	88.501%	88.322%	88.229%	88.184%	88.108%
7	81.682%	81.682%	81.682%	81.682%	81.787%	81.724%	81.692%	81.682%	81.682%	81.692%
8	82.000%	83.748%	82.612%	82.123%	84.181%	83.970%	83.305%	81.958%	82.844%	82.413%
9	80.566%	80.850%	80.541%	80.708%	80.775%	80.686%	80.835%	80.768%	80.592%	80.568%
10	78.629%	79.010%	78.480%	78.352%	78.925%	78.757%	78.757%	78.383%	78.531%	78.503%
Average	84.678%	84.923%	84.683%	84.637%	85.112%	85.058%	84.936%	84.596%	84.716%	84.679%
s1c5b10	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	80.217%	80.726%	80.230%	80.430%	80.776%	80.682%	80.464%	80.206%	80.292%	80.254%
2	89.859%	90.002%	89.162%	89.420%	91.891%	91.742%	90.966%	88.982%	90.199%	88.980%
3	72.649%	72.629%	72.630%	72.516%	72.678%	72.792%	72.796%	72.284%	72.435%	72.633%
4	74.553%	75.209%	74.961%	74.231%	75.260%	75.471%	75.079%	74.856%	74.965%	75.188%
5	81.947%	82.122%	81.585%	81.941%	81.892%	81.938%	81.652%	81.392%	81.750%	81.761%
6	88.131%	87.898%	88.328%	88.076%	89.925%	89.418%	89.056%	88.103%	88.046%	88.300%
7	86.605%	86.804%	86.636%	86.779%	87.003%	87.349%	86.908%	86.381%	86.617%	86.649%
8	84.348%	84.348%	84.376%	84.582%	84.376%	84.376%	84.109%	84.037%	84.435%	84.435%
9	92.104%	92.172%	91.760%	91.749%	92.259%	92.179%	92.294%	92.346%	91.937%	91.900%
10	93.648%	93.706%	93.648%	93.648%	93.703%	93.657%	93.648%	93.648%	93.648%	93.648%
Average	84.406%	84.562%	84.332%	84.337%	84.976%	84.960%	84.697%	84.223%	84.432%	84.375%
s1c5b25	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	81.002%	81.150%	81.002%	81.002%	81.120%	81.237%	81.118%	80.887%	81.002%	81.002%
2	83.879%	83.984%	83.879%	83.879%	83.858%	83.822%	83.841%	83.742%	83.879%	83.890%
3	87.564%	87.857%	87.474%	87.412%	88.539%	88.285%	88.049%	87.491%	87.738%	87.615%
4	85.324%	85.566%	84.824%	84.629%	85.838%	86.139%	86.427%	84.643%	85.148%	84.909%
5	82.021%	82.384%	82.603%	82.541%	83.530%	83.015%	82.707%	81.918%	82.027%	82.302%
6	85.360%	85.463%	85.529%	85.435%	85.360%	85.343%	85.415%	85.335%	85.335%	85.511%
7	82.921%	83.228%	82.821%	82.862%	83.564%	83.564%	82.922%	82.754%	82.965%	83.026%
8	87.568%	87.568%	86.452%	86.570%	87.610%	87.727%	88.694%	86.646%	87.568%	87.568%
9	91.600%	91.600%	91.847%	91.633%	91.900%	92.053%	92.327%	91.534%	91.600%	91.600%
10	87.510%	87.438%	87.328%	87.963%	88.251%	88.464%	87.813%	87.595%	87.438%	87.769%
Average	85.475%	85.624%	85.376%	85.393%	85.957%	85.965%	85.931%	85.254%	85.470%	85.519%

Table 5.6 Results for ILS with Simple Neighbourhood on data s1c5

The result under experiment allBoxes comes as the second best and the result under experiment 11Column is just 0.03% better than the one under experiment allBoxes. Data set s2c10b25 has its best result generated under experiment wall. However, the result under experiment wall is just 0.01% better than the one under experiment allBoxes.

The cost of running Cargo Manager is proved to be incredible high, as time spent on executing CM is at least 99.9% of overall running time for both soton1 and soton2. For results run under allBoxes experiment, the number of CM executed is 3269, 2381, 682, 3307, 2634 and 1961 for data s2c5b5, s2c5b10, s2c5b25, s2c10b5, s2c10b10 and s2c10b25 respectively. However, the standard deviation for s2c5b25 is 306 and that of s2c10b25 is 1008. The number of CM executed is just over 200 for some instances in those two data sets. The reason behind this is that Cargo Manager has a hard time to deal with data with over 10,000 boxes together with a high box types such as 25 as explained in section 4.5.1. The number of kicks

s1c10b5	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	82.293%	82.850%	82.554%	82.445%	82.866%	82.797%	82.615%	82.191%	82.361%	82.519%
2	86.928%	86.849%	87.081%	87.083%	87.115%	87.393%	87.568%	86.929%	87.138%	87.155%
3	76.981%	77.101%	76.591%	76.170%	79.835%	79.651%	79.197%	77.472%	76.762%	77.390%
4	72.605%	73.063%	72.542%	72.416%	72.741%	72.614%	72.531%	72.480%	72.607%	72.545%
5	76.767%	76.995%	76.785%	76.914%	76.920%	76.920%	76.995%	76.635%	76.739%	76.767%
6	84.578%	84.710%	84.357%	84.363%	84.471%	84.536%	84.307%	84.356%	84.321%	84.537%
7	88.178%	88.460%	88.307%	88.443%	89.168%	88.828%	88.715%	88.036%	88.386%	88.255%
8	75.221%	75.853%	74.051%	72.790%	75.834%	75.700%	74.452%	75.065%	74.640%	75.222%
9	76.662%	76.811%	76.605%	76.581%	77.193%	77.467%	77.125%	76.416%	76.811%	76.679%
10	76.516%	76.531%	76.153%	76.300%	76.725%	76.634%	76.317%	76.341%	76.211%	76.373%
Average	79.673%	79.922%	79.503%	79.350%	80.287%	80.254%	79.982%	79.592%	79.598%	79.744%
s1c10b10	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	80.649%	81.137%	80.476%	80.515%	81.034%	81.022%	80.961%	80.437%	80.323%	80.561%
2	82.434%	82.365%	82.393%	82.264%	82.530%	82.587%	82.815%	82.173%	82.374%	82.233%
3	79.370%	79.438%	79.392%	79.365%	79.359%	79.372%	79.359%	79.359%	79.359%	79.361%
4	76.513%	76.872%	76.400%	76.414%	76.722%	76.674%	76.538%	76.460%	76.463%	76.396%
5	85.837%	85.383%	84.850%	84.889%	85.706%	85.484%	84.974%	85.550%	85.394%	85.935%
6	89.774%	90.157%	89.829%	90.152%	90.172%	90.170%	90.431%	89.254%	89.735%	90.010%
7	87.531%	88.568%	87.282%	85.855%	88.742%	88.640%	87.983%	86.950%	87.042%	87.448%
8	73.551%	73.635%	73.529%	73.624%	73.933%	74.013%	73.676%	73.409%	73.628%	73.451%
9	80.906%	81.021%	80.848%	81.002%	81.279%	81.346%	81.257%	80.862%	80.926%	81.024%
10	84.647%	84.704%	84.647%	84.619%	84.652%	84.732%	84.676%	84.619%	84.647%	84.647%
Average	82.121%	82.328%	81.964%	81.870%	82.413%	82.404%	82.267%	81.907%	81.989%	82.107%
s1c10b25	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	81.660%	82.969%	81.403%	81.210%	83.528%	81.982%	81.877%	81.980%	82.001%	81.785%
2	89.962%	90.091%	89.299%	89.275%	89.797%	89.798%	89.392%	89.287%	89.962%	89.962%
3	79.067%	79.579%	79.299%	79.209%	79.967%	79.813%	79.337%	79.260%	79.080%	79.426%
4	82.353%	82.489%	82.457%	82.157%	83.564%	83.601%	82.846%	82.864%	82.415%	82.251%
5	79.395%	79.812%	79.531%	79.469%	80.356%	80.119%	79.661%	79.286%	79.530%	79.772%
6	82.160%	82.881%	81.008%	81.219%	82.538%	82.158%	81.730%	81.899%	81.141%	82.043%
7	82.384%	82.663%	82.263%	82.067%	82.912%	82.826%	82.775%	82.170%	82.164%	82.544%
8	76.960%	77.518%	77.057%	77.022%	78.223%	77.779%	77.706%	77.012%	77.826%	77.120%
9	89.532%	89.688%	89.819%	89.363%	90.329%	90.117%	89.762%	89.290%	89.748%	89.715%
10	84.452%	84.755%	84.435%	84.539%	85.625%	85.759%	84.965%	84.306%	84.467%	84.306%
Average	82.792%	83.245%	82.657%	82.553%	83.684%	83.395%	83.005%	82.735%	82.833%	82.892%

Table 5.7 Results for ILS with Simple Neighbourhood on data s1c10

is 28.66 (with a standard deviation of 4.32), 14.88 (24.68), 0.00, 12.38 (2.75), 2.74 (0.99) and 0.46 (0.65) for data s2c5b5, s2c5b10, s2c5b25, s2c10b5, s2c10b10 and s2c10b25 respectively. The standard deviation for s2c5b10 suggests that kick does not happen in some instances within that data cluster. Also, the numbers of kicks in s2c5b25, s2c10b10 and s2c10b25 indicate that search never really expands into other neighbourhoods. To sum up, results run under experiment allBoxes are better than those run under other conditions for both data set soton1 and soton2.

Table 5.10 summarizes the initial solution, and solutions generated by ILSIN (default sequencing), by ILSDR (default sequencing), and by ILSSN (allBoxes) for data set soton1. Initial solutions are generated by executing CM repeatedly till a list of pre-ordered boxes are all packed. Generating initial solutions is a necessary step for ILSSN but not for ILSIN or ILSDR as those two algorithms only generate partial initial solution at beginning. Solutions

s2c5b5	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	93.893%	94.004%	93.904%	93.920%	94.017%	93.954%	94.147%	93.635%	93.926%	94.036%
2	92.376%	92.401%	92.375%	92.375%	92.490%	92.453%	92.451%	92.346%	92.372%	92.366%
3	90.183%	90.392%	90.177%	90.119%	90.374%	90.238%	90.215%	90.094%	90.078%	90.148%
4	91.291%	91.497%	91.250%	91.209%	91.502%	91.371%	91.398%	91.140%	91.283%	91.267%
5	91.901%	91.878%	91.972%	91.933%	91.996%	91.922%	92.076%	91.727%	92.004%	91.926%
6	93.450%	93.664%	93.441%	93.442%	93.779%	93.566%	93.596%	93.294%	93.466%	93.459%
7	95.296%	95.348%	95.296%	95.296%	95.417%	95.318%	95.320%	95.296%	95.296%	95.296%
8	93.258%	93.432%	93.225%	93.229%	93.675%	93.319%	93.368%	93.225%	93.297%	93.225%
9	88.474%	88.594%	88.497%	88.498%	88.650%	88.551%	88.524%	88.456%	88.513%	88.510%
10	94.298%	94.337%	94.298%	94.298%	94.444%	94.298%	94.322%	94.298%	94.298%	94.298%
Average	92.442%	92.555%	92.444%	92.432%	92.634%	92.499%	92.542%	92.351%	92.453%	92.453%
s2c5b10	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	93.007%	92.956%	92.842%	92.671%	93.296%	93.182%	92.934%	92.631%	92.981%	92.889%
2	94.880%	94.994%	94.878%	94.878%	95.017%	94.878%	94.896%	94.878%	94.878%	94.878%
3	79.624%	79.624%	79.624%	79.624%	79.624%	79.624%	79.624%	79.624%	79.624%	79.624%
4	94.088%	94.093%	94.088%	94.088%	94.375%	94.176%	94.192%	94.102%	94.088%	94.094%
5	92.550%	92.088%	92.323%	92.366%	94.018%	93.221%	92.964%	91.943%	92.797%	92.527%
6	94.484%	94.484%	94.484%	94.484%	94.944%	94.534%	94.544%	94.484%	94.484%	94.484%
7	92.854%	92.864%	92.763%	92.763%	93.252%	93.199%	93.299%	92.724%	92.963%	92.943%
8	95.132%	95.304%	95.155%	95.057%	95.632%	95.330%	95.551%	94.908%	95.212%	95.118%
9	95.767%	95.770%	95.765%	95.765%	95.783%	95.776%	95.784%	95.776%	95.765%	95.765%
10	93.400%	93.707%	93.361%	93.431%	94.207%	93.865%	93.762%	93.307%	93.569%	93.615%
Average	92.579%	92.588%	92.528%	92.513%	93.015%	92.779%	92.755%	92.438%	92.636%	92.594%
s2c5b25	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	90.546%	90.546%	90.546%	90.546%	90.546%	90.546%	90.546%	90.546%	90.546%	90.964%
2	93.810%	93.810%	93.810%	93.810%	93.810%	93.810%	93.810%	93.815%	93.810%	93.810%
3	94.155%	94.155%	94.155%	94.155%	94.480%	94.377%	94.107%	94.147%	94.155%	94.155%
4	96.177%	96.177%	96.172%	96.172%	96.139%	96.139%	96.139%	96.139%	96.177%	96.314%
5	94.168%	94.168%	94.168%	94.043%	94.376%	94.233%	94.201%	94.168%	94.184%	94.184%
6	94.438%	94.438%	94.438%	94.438%	94.438%	94.438%	94.438%	94.438%	94.438%	94.438%
7	93.035%	93.035%	93.035%	93.027%	93.027%	93.027%	93.027%	93.027%	93.035%	93.035%
8	91.323%	91.323%	91.323%	91.323%	91.237%	91.342%	91.362%	91.408%	91.323%	91.323%
9	93.745%	93.745%	93.745%	93.745%	93.745%	94.326%	94.209%	93.745%	93.745%	93.745%
10	93.141%	93.229%	93.271%	93.158%	93.911%	93.755%	93.595%	93.216%	93.271%	93.271%
Average	93.454%	93.463%	93.466%	93.442%	93.571%	93.599%	93.543%	93.465%	93.468%	93.524%

Table 5.8 Results for ILS with Simple Neighbourhood on data s2c5

generated by ILSSN (allBoxes) are better in general. For data s1c5b5, s1c5b10, s1c5b25, s1c10b5, s1c10b10 and s1c10b25, solutions generated by ILSSN (allBoxes) are 2.74%, 2.25%, 1.75%, 2.31%, 1.67% and 2.26% better than the initial solutions. While solutions generated by ILSIN (default) come second except for s1c5b5, those generated by ILSSN (allBoxes) are 1.54%, 0.89%, 1.04%, 0.54%, 1.18% and 0.97% better when compared to ILSIN. Solutions generated by ILSIN (default) are 0.84%, 0.56%, 0.29%, 0.12% and 0.78% better than those generated by ILSDR for data s1c5b10, s1c5b25, s1c10b5, s1c10b10 and s1c10b25. For data s1c5b5 solution generated by ILSDR is 0.62% better than ILSIN's. Solutions generated by three algorithms all have improvements compared to the initial solutions.

Table 5.11 summarizes the initial solution, and solutions generated by ILSIN (default sequencing), by ILSDR (default sequencing), and by ILSSN (allBoxes) for soton2. Again,

s2c10b5	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	95.384%	95.384%	95.384%	95.384%	95.365%	95.336%	95.331%	95.331%	95.384%	95.384%
2	93.828%	93.828%	93.828%	93.828%	93.828%	93.828%	93.828%	93.828%	93.828%	93.828%
3	94.446%	94.616%	94.479%	94.481%	94.667%	94.491%	94.560%	94.455%	94.540%	94.488%
4	95.134%	95.239%	95.117%	95.170%	95.763%	95.358%	95.448%	95.065%	95.225%	95.121%
5	94.626%	94.649%	94.613%	94.613%	94.653%	94.638%	94.619%	94.613%	94.613%	94.630%
6	93.825%	93.940%	93.721%	93.766%	93.918%	94.001%	94.025%	93.709%	93.758%	93.809%
7	90.993%	91.011%	91.013%	90.935%	91.150%	91.004%	91.157%	90.824%	90.935%	90.992%
8	95.329%	95.328%	95.313%	95.281%	95.718%	95.404%	95.344%	95.272%	95.338%	95.284%
9	91.573%	91.628%	91.688%	91.591%	91.884%	91.734%	91.890%	91.530%	91.635%	91.634%
10	94.424%	94.204%	94.303%	94.354%	95.541%	94.947%	95.250%	93.724%	94.148%	94.329%
Average	93.956%	93.983%	93.946%	93.940%	94.249%	94.074%	94.145%	93.835%	93.940%	93.950%
s2c10b10	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	94.082%	94.069%	94.067%	94.025%	94.124%	94.077%	94.061%	94.045%	94.067%	94.069%
2	94.031%	94.027%	94.061%	94.025%	94.021%	94.018%	94.040%	94.032%	94.029%	94.026%
3	95.850%	95.903%	95.845%	95.845%	96.277%	96.124%	96.123%	95.845%	95.845%	95.845%
4	95.783%	95.862%	95.783%	95.783%	96.279%	95.802%	95.822%	95.783%	95.854%	95.848%
5	95.554%	95.571%	95.674%	95.674%	95.709%	95.552%	95.580%	95.545%	95.546%	95.546%
6	94.601%	94.645%	94.550%	94.540%	94.768%	94.590%	94.665%	94.506%	94.561%	94.589%
7	94.076%	94.099%	94.050%	94.044%	94.272%	94.167%	94.137%	94.063%	94.086%	94.076%
8	91.571%	91.728%	91.599%	91.656%	92.167%	91.665%	91.815%	91.497%	91.537%	91.539%
9	94.887%	94.918%	94.911%	94.878%	95.990%	95.080%	95.587%	94.854%	94.904%	94.878%
10	96.493%	96.499%	96.496%	96.483%	97.031%	96.587%	96.707%	96.470%	96.489%	96.519%
Average	94.693%	94.732%	94.704%	94.695%	95.064%	94.766%	94.854%	94.664%	94.692%	94.693%
s2c10b25	default	fTCM	totalBoxVolume	boxNumber	allBoxes	11Column	wall	oneBox	ico	endIco
1	94.461%	94.513%	94.546%	94.439%	94.386%	94.431%	94.386%	94.385%	94.581%	94.581%
2	93.717%	93.717%	93.690%	93.667%	93.728%	93.615%	93.803%	93.669%	93.717%	93.717%
3	94.223%	94.223%	94.223%	94.223%	94.223%	94.223%	94.223%	94.223%	94.223%	94.223%
4	93.481%	93.614%	93.458%	93.430%	93.747%	93.633%	93.763%	93.421%	93.531%	93.476%
5	92.845%	92.845%	92.856%	92.789%	92.958%	92.878%	92.891%	92.841%	92.845%	92.845%
6	94.039%	94.039%	94.039%	94.088%	93.868%	93.868%	93.865%	93.865%	94.039%	94.052%
7	79.359%	79.631%	79.388%	79.359%	79.550%	79.518%	79.602%	79.449%	79.359%	79.359%
8	91.935%	91.935%	91.935%	91.950%	91.922%	91.922%	91.922%	91.935%	91.935%	91.935%
9	93.616%	93.641%	93.496%	93.468%	93.751%	93.746%	93.751%	93.478%	93.651%	93.760%
10	94.011%	94.011%	94.011%	94.011%	94.011%	94.011%	94.011%	94.011%	94.011%	94.149%
Average	92.169%	92.217%	92.164%	92.142%	92.214%	92.184%	92.222%	92.128%	92.189%	92.210%

Table 5.9 Results for ILS with Simple Neighbourhood on data s2c10

solutions generated by ILSSN (allBoxes) are better in general. One other observation is that neither solution generated by ILSIN or ILSDR is better than the initial solution for s2c10b5. One of the reasons of that all search algorithms achieve less on data set soton2 is that boxes in soton2 are much smaller compared to those in soton1 and they already are tightly packed in nature leaving less space for later improvement.

In summary, almost all solutions generated by three iterated local search algorithms have improved when comparing to initial solutions generated by iterated local search with simple neighbourhood (ILSSN). More significant improvements are observed for data set soton1 compared to those of data set soton2. One cause for this is that smaller boxes in soton2 have already been more tightly packed thus less space is left for future improvement. Overall, ILSSN generates better results than ILSIN and ILSDR. This implies that a goal-oriented search frame with simple neighbourhood move works better when we have little control

Instance	s1c5b5				s1c5b10				s1c5b25			
	Initial	ILSSN	ILSIN	ILSDR	Initial	ILSSN	ILSIN	ILSDR	Initial	ILSSN	ILSIN	ILSDR
1	87.299%	87.967%	87.299%	87.403%	80.206%	80.776%	81.064%	80.275%	80.253%	81.120%	81.398%	81.027%
2	71.152%	72.685%	72.421%	72.677%	87.656%	91.891%	89.277%	88.346%	83.742%	83.858%	83.742%	83.682%
3	91.872%	92.353%	91.872%	91.872%	71.834%	72.678%	71.545%	70.915%	86.802%	88.539%	86.984%	86.802%
4	90.114%	91.566%	90.747%	90.186%	73.856%	75.260%	74.826%	74.658%	83.566%	85.838%	84.454%	84.048%
5	91.972%	92.234%	91.972%	92.068%	78.707%	81.892%	80.176%	79.944%	81.103%	83.530%	82.428%	81.103%
6	88.108%	88.643%	88.108%	88.392%	84.201%	89.925%	89.586%	86.270%	84.309%	85.360%	84.751%	84.309%
7	81.682%	81.787%	81.599%	81.634%	86.216%	87.003%	86.342%	86.216%	81.117%	83.564%	81.729%	81.028%
8	69.114%	84.181%	75.281%	80.689%	83.210%	84.376%	84.167%	83.548%	85.546%	87.610%	86.287%	85.599%
9	80.076%	80.775%	80.646%	80.348%	91.511%	92.259%	91.671%	91.511%	91.159%	91.900%	91.644%	91.159%
10	77.013%	78.925%	78.297%	78.167%	93.648%	93.703%	93.648%	93.648%	87.200%	88.251%	87.315%	87.200%
Average	82.840%	85.112%	83.824%	84.344%	83.105%	84.976%	84.230%	83.533%	84.480%	85.957%	85.073%	84.596%

Instance	s1c10b5				s1c10b10				s1c10b25			
	Initial	ILSSN	ILSIN	ILSDR	Initial	ILSSN	ILSIN	ILSDR	Initial	ILSSN	ILSIN	ILSDR
1	82.172%	82.866%	82.715%	82.690%	79.850%	81.034%	80.362%	80.166%	80.828%	83.528%	81.462%	80.967%
2	86.578%	87.115%	87.011%	86.830%	81.804%	82.530%	81.978%	81.977%	89.162%	89.797%	89.162%	89.162%
3	75.661%	79.835%	80.149%	77.943%	79.359%	79.359%	77.398%	77.946%	77.636%	79.967%	80.322%	78.692%
4	71.907%	72.741%	72.247%	72.572%	75.925%	76.722%	76.876%	76.801%	81.152%	83.564%	82.862%	81.436%
5	76.208%	76.920%	76.995%	76.621%	84.638%	85.706%	85.396%	85.282%	78.378%	80.356%	79.638%	78.802%
6	84.083%	84.471%	84.083%	84.087%	88.433%	90.172%	89.569%	88.810%	79.364%	82.538%	82.664%	81.243%
7	87.833%	89.168%	88.029%	88.285%	84.122%	88.742%	84.362%	85.473%	81.750%	82.912%	81.873%	81.746%
8	69.084%	75.834%	75.137%	73.998%	72.705%	73.933%	74.019%	73.043%	76.577%	78.223%	76.824%	76.823%
9	75.637%	77.193%	75.799%	76.467%	79.158%	81.279%	79.927%	79.424%	89.202%	90.329%	89.693%	89.202%
10	75.595%	76.725%	76.392%	76.798%	84.619%	84.652%	84.619%	84.606%	84.306%	85.625%	84.339%	84.377%
Average	78.476%	80.287%	79.856%	79.629%	81.061%	82.413%	81.450%	81.353%	81.835%	83.684%	82.884%	82.245%

Table 5.10 soton1 Results Comparison among Initial Solution, ILS with Simple Neighbourhood, ILS with Intelligent Neighborhood, and ILS with Descent and Randomness

over constructive heuristic. For ILSIN and ILSDR, partial initial solution generated by default sequencing generally leads to better final solution. For ILSSN, experiment allBoxes generates better solutions. Running Cargo Manager is proved to be incredibly costly as over 99% of the overall running time is spent on executing CM. On the other hand, our evaluation design, neighbourhood evaluation in ILSIN and split batch in ILSDR, is not time-consuming. As we implemented the strategy of split batch in both ILSIN and ILSDR, positive impact is observed in ILSIN while randomly split batch performs mediocly in ILSDR. For ILSIN and ILSSN, fewer kicks are observed for data sets having more box types and requiring more containers. Kicks get fewer and fewer especially for data set soton2 with more box types and container requirement as over 10,000 boxes together with higher number of box types require Cargo Manager to spend significant more time on each run as shown in section 4.5.1.

5.6.3 Beam Search Results under Different Parameters

In our beam search design, we have a few parameters including filter width α , beam width β , and the number(s) of child nodes generated under Strategy 1 to 6. In this section, we investigate the impact of different parameters upon the performance of our algorithm. We

Instance	s2c5b5				s2c5b10				s2c5b25			
	Initial	ILSSN	ILSIN	ILSDR	Initial	ILSSN	ILSIN	ILSDR	Initial	ILSSN	ILSIN	ILSDR
1	93.478%	94.017%	93.237%	93.580%	92.616%	93.296%	92.475%	92.434%	90.546%	90.546%	90.692%	90.605%
2	92.120%	92.490%	92.607%	92.607%	94.878%	95.017%	94.720%	94.769%	93.810%	93.810%	94.384%	94.383%
3	89.831%	90.374%	90.450%	90.378%	79.624%	79.624%	80.460%	80.460%	94.107%	94.377%	94.345%	94.345%
4	90.589%	91.502%	90.599%	90.881%	94.088%	94.375%	93.954%	93.954%	96.139%	96.139%	95.766%	95.766%
5	91.390%	91.996%	91.434%	91.493%	90.850%	94.018%	91.450%	91.939%	94.020%	94.233%	94.128%	94.128%
6	93.098%	93.779%	94.028%	94.028%	94.484%	94.944%	94.455%	94.500%	94.438%	94.438%	94.457%	94.438%
7	95.296%	95.417%	95.310%	95.342%	92.724%	93.252%	92.627%	92.636%	93.027%	93.027%	93.125%	92.914%
8	93.061%	93.675%	93.333%	93.391%	94.831%	95.632%	93.727%	93.727%	91.237%	91.342%	90.722%	90.798%
9	88.437%	88.650%	88.586%	88.586%	95.765%	95.783%	95.629%	95.629%	93.745%	94.326%	94.140%	94.140%
10	94.298%	94.444%	93.174%	93.268%	93.138%	94.207%	93.541%	93.541%	92.953%	93.755%	92.467%	92.480%
Average	92.160%	92.634%	92.276%	92.355%	92.300%	93.015%	92.304%	92.359%	93.402%	93.599%	93.423%	93.400%
Instance	s2c10b5				s2c10b10				s2c10b25			
	Initial	ILSSN	ILSIN	ILSDR	Initial	ILSSN	ILSIN	ILSDR	Initial	ILSSN	ILSIN	ILSDR
1	95.331%	95.365%	95.506%	95.506%	93.979%	94.124%	94.269%	94.269%	94.357%	94.386%	94.094%	94.094%
2	93.623%	93.828%	93.664%	93.667%	93.940%	94.021%	93.691%	93.691%	93.615%	93.803%	93.537%	93.537%
3	93.908%	94.667%	94.259%	94.318%	95.845%	96.277%	95.972%	95.888%	94.223%	94.223%	93.148%	93.148%
4	95.029%	95.763%	95.087%	94.831%	95.783%	96.279%	94.810%	94.810%	93.421%	93.763%	93.027%	93.015%
5	94.613%	94.653%	91.608%	92.175%	95.535%	95.709%	95.733%	95.732%	92.783%	92.891%	92.628%	92.628%
6	93.415%	93.918%	93.809%	93.878%	94.476%	94.768%	94.810%	94.799%	93.865%	93.865%	94.084%	94.055%
7	90.693%	91.150%	90.586%	90.857%	93.869%	94.272%	94.117%	94.117%	79.024%	79.602%	80.973%	80.394%
8	95.251%	95.718%	95.485%	95.522%	91.346%	92.167%	91.960%	92.023%	91.922%	91.922%	92.350%	92.185%
9	91.372%	91.884%	91.634%	91.578%	94.854%	95.990%	95.157%	94.982%	93.297%	93.751%	94.054%	94.054%
10	93.524%	95.541%	93.618%	94.121%	96.470%	97.031%	96.460%	96.466%	94.011%	94.011%	93.153%	93.153%
Average	93.676%	94.249%	93.526%	93.645%	94.610%	95.064%	94.698%	94.678%	92.052%	92.222%	92.105%	92.026%

Table 5.11 soton2 Results Comparison among Initial Solution, ILS with Simple Neighbourhood, ILS with Intelligent Neighborhood, and ILS with Descent and Randomness

experiment using data sets s1c5b5, s1c5b10, and s1c5b25. We consider these three data sets as one close group and our in-sample for beam search, and carry on experiment using the same parameters. The rest data sets, namely s1c10, s2c5 and s2c10, are effectively our out of sample for beam search. The performance of our algorithm is reported in terms of percentage utilisation same as for Iterated Local Search, so comparison of performance can be drawn between these two algorithms in the next section. It is worth pointing out that while we set the same parameters for all three data sets, we aim at 500 seconds running time for one run of beam search approach. However, since we can only estimate approximately how long a single run will take under the parameters we set, it is almost certain that a single run will not be exact 500 seconds. Because the results of Iterated Local Search are generated under 500 seconds, a fair comparison between these two algorithms is hard to be drawn. Moreover, we already learned that data sets having more box types need more time to run from section 4.5.1. Such fact indicates that if one run of s1c5b10 spends exact 500 seconds, one run of s1c5b25 will probably spend more than 500 seconds and one run of s1c5b5 will spend less than 500 seconds both by a noticeable time margin. Since this is a matter of fact and we have to run all data sets within a close group with same parameters, our aim is to run data sets with 10 box types for as close as 500 seconds and to accept that data sets with 5 box types will have a shorter running time and a longer running time for data sets with 25 box types.

Therefore, we design our parameter investigation as follows. Firstly we investigate the impact of filter width and beam width while keeping the number of child nodes unchanged. We are aware of the possibility of having all next-level beams branched from one single beam when filter width is set as a high value and beam width as a low value. Therefore, we initially set filter width much smaller than beam width to make sure that the next-level beams will be branched from various beams. Then we adjust beam width and the number of child nodes while keeping filter width unchanged. Finally, we change filter width and the number of child nodes while beam width remains the same.

Table 5.12 compares the results and average computation times under different parameters for data set s1c5b5. Each instance is run 5 times and the average result is taken. The average of computation times for all 10 instances is also recorded. The best results and computation time have been highlighted in bold. For the first three columns we set the number of child nodes generated as 10 for Strategy 1 to 6. As Strategy 0 always generates 2 child nodes, that means a total of 62 child nodes are generated from a beam node. We firstly set filter width = 5 and beam width = 20 as the experiment named [f5 b20 stra10]. We then set filter width = 10 and beam width = 15 [f10 b15 stra10]. Finally we increase filter width to 30 and reduce beam width to 10 [f30 b10 stra10].

Among the above three settings, the same number of child nodes guarantees the quality of available pool of partial solutions where filter nodes can be chosen from. While [f5 b20 stra10] ensures the search chooses the next-level beam nodes from different beams, it chooses the greediest filter nodes when compared to [f30 b10 stra10]. The results show that [f30 b10 stra10] generates better results among the three. While it has the highest computation time of 542 seconds, the time margin compared to the other two is not considered significant enough to significantly impact the results. Therefore, we carry out two more experiments based on [f30 b10 stra10].

We firstly increase the number of child nodes to 28 and reduce beam width to 5 [f30 b5 stra28] while keeping filter width unchanged. This move potentially improves the quality of filtered nodes by increasing the number of child nodes while only those evaluated better globally are chosen. Our other experiment increases the number of child nodes to 20 and reduces filter width to 5 while beam width remains the same as 10 [f5 b10 stra20]. This setting potentially increases the overall quality of child nodes however they will be pruned more greedily. Results show that [f30 b10 stra10] still has the best results with a tiny time margin which is insignificant.

Same parameter settings are implemented on data sets s1c5b10 and s1c5b25. Their results are shown in the Table 5.13 and Table 5.14 respectively. Parameter setting [f30 b10 stra10]

Instance	s1c5b5				
	f5 b20 stra10	f10 b15 stra10	f30 b10 stra10	f30 b5 stra28	f5 b10 stra20
1	87.817%	87.910%	88.090%	87.727%	87.865%
2	72.021%	72.194%	73.518%	72.280%	71.367%
3	92.123%	92.421%	92.438%	92.511%	92.349%
4	92.112%	92.335%	92.150%	92.205%	91.955%
5	92.832%	92.944%	92.927%	93.130%	92.675%
6	88.817%	89.173%	89.238%	89.213%	89.185%
7	81.839%	81.945%	81.971%	81.950%	81.683%
8	82.773%	82.659%	83.715%	74.594%	82.642%
9	81.161%	81.224%	81.278%	81.290%	81.279%
10	79.296%	79.249%	79.249%	79.173%	79.708%
Ave	85.079%	85.205%	85.457%	84.407%	85.071%
Run Time (s)	527	498	542	535	533

Table 5.12 Beam Search Results under Different Parameters for s1c5b5

generates the best results for data set s1c5b10 with the least computation time used. For data set s1c5b25, [f30 b10 stra10] does not generate the best results. However, it has the second least computation time of 934 seconds, which is only 2 seconds more than the least computation time spent by [f10 b15 stra10]. Parameter setting [f30 b5 stra28] generates the best results with second most computation time of 1053 seconds, which is 119 seconds more than that of [f30 b10 stra10]. Overall, results under different parameters for s1c5b10 and s1c5b25 do not have huge difference among themselves. While the high cost of Cargo Manager (CM)'s computation time significantly limited the scale of our experiment, we attempt to draw a couple of clearer observations. Firstly, it may not be a good idea to over-greedily filter child nodes by setting a considerably small filter width. Also, the intention of avoiding completely pruning one or more beam may not improve the results, at least when it is done by over-greedily setting small filter width.

5.6.4 Control-Set and Results Comparison

In this session, we compare the results of beam search with initial solutions and those of iterated local search. Also, we are well aware of that algorithms might not be actually doing work even when they generate improved results. Therefore, we design a 'Control-Set' to check if both of our algorithms really work. The idea of 'Control-Set' is to randomly pack containers for 500 seconds, and compare it with results of both iterated local search and beam search. If the results of algorithms do not have advantage over that of 'Control-Set',

Instance	s1c5b10				
	f5 b20 stra10	f10 b15 stra10	f30 b10 stra10	f30 b5 stra28	f5 b10 stra20
1	81.092%	81.334%	81.577%	81.538%	81.449%
2	92.772%	92.829%	92.807%	92.883%	92.748%
3	72.945%	73.393%	73.454%	73.494%	73.286%
4	75.899%	75.988%	75.983%	76.064%	76.022%
5	81.792%	82.073%	82.876%	82.295%	81.796%
6	91.456%	91.619%	91.411%	91.559%	91.498%
7	87.494%	87.784%	88.149%	87.957%	87.753%
8	85.271%	85.402%	86.035%	85.450%	84.948%
9	93.040%	93.196%	93.145%	93.378%	93.235%
10	94.323%	94.204%	94.123%	94.181%	94.389%
Ave	85.608%	85.782%	85.956%	85.880%	85.712%
Run Time (s)	553	526	504	565	578

Table 5.13 Beam Search Results under Different Parameters for s1c5b10

Instance	s1c5b25				
	f5 b20 stra10	f10 b15 stra10	f30 b10 stra10	f30 b5 stra28	f5 b10 stra20
1	81.171%	81.565%	81.709%	81.709%	81.489%
2	83.687%	84.166%	84.360%	84.373%	84.094%
3	88.784%	88.868%	88.963%	89.256%	89.127%
4	86.899%	87.362%	87.413%	87.746%	87.310%
5	83.356%	83.553%	83.328%	83.532%	83.574%
6	85.837%	86.338%	86.335%	86.761%	86.351%
7	84.502%	85.048%	84.569%	84.370%	84.217%
8	87.991%	87.803%	87.718%	88.242%	88.441%
9	92.712%	92.789%	92.647%	92.907%	92.829%
10	87.910%	88.105%	87.986%	88.058%	88.185%
Ave	86.285%	86.560%	86.503%	86.695%	86.562%
Run Time (s)	990	932	934	1053	1087

Table 5.14 Beam Search Results under Different Parameters for s1c5b25

then the algorithm(s) is not considered as working well. Our ‘Control-Set’ implementation places all boxes within their original batch, and sort these original box types into a random sequencing. It then repeatedly runs Cargo Manager (CM) until all boxes are packed. The average utilisation is calculated the same way as in the Iterated Local Search and Beam Search algorithms. The same process is repeated for 500 seconds, and the best average utilisation is selected as the result. Each instance is run 5 times, and the average of these five results is taken as the final result of ‘Control-Set’.

Table 5.15 compares the results among initial solution, Control-Set, Iterated Local Search (ILS) and Beam Search (BS) for data sets soton1. Results are presented in terms of solution quality as utilisation. The best results are highlighted in bold. The computation times of beam search are reported as well. We pick Iterated Local Search with Simple Neighbourhood (ILSSN) to represent iterated local search as it generates the overall best results. For data sets s1c5b5, s1c5b10 and s1c5b25, the results of parameter setting [f30 b10 stra10] is presented for beam search. For data sets s1c10b5, s1c10b10 and s1c10b25, we set parameters as [f15 b5 stra5]. We set parameters based on the observation in Section 5.6.3, and ideally one run of beam search will have a computation time of 500 seconds.

For data set s1c5b5, beam search generates the best overall results except for instance 8 where Control-Set beats both algorithms. The overall improvement rates over initial solutions are 2.244%, 2.742% and 3.159% for Control-Set, ILS and BS respectively. The average computation time for BS is 542 seconds. Similar outcomes are observed for other five data sets, i.e. s1c5b10, s1c5b25, s1c10b5, s1c10b10 and s1c10b25, with beam search outperforming others on all averages (and for almost all instances). In general, the results of ILSSN come second and those of ‘Control-Set’ come third. ‘Control-Set’ only fails to improve with its average result 0.025% worse than initial solution for data set s1c5b25.

Instance	s1c5b5				s1c5b10				s1c5b25			
	Initial	Control-Set	Local Search ILSSN	Beam Search f30 b10 stra10 Run Time (s)	Initial	Control-Set	Local Search ILSSN	Beam Search f30 b10 stra10 Run Time (s)	Initial	Control-Set	Local Search ILSSN	Beam Search f30 b10 stra10 Run Time (s)
1	87.299%	87.299%	87.967%	463	80.206%	80.349%	80.776%	81.577%	80.253%	80.004%	81.120%	81.709%
2	71.152%	72.272%	72.685%	633	87.656%	91.860%	91.891%	92.807%	83.742%	83.325%	83.858%	84.360%
3	91.872%	91.872%	92.353%	460	71.834%	72.455%	72.678%	73.454%	86.802%	85.985%	88.539%	88.963%
4	90.114%	91.165%	91.566%	443	73.856%	75.153%	75.260%	75.983%	83.566%	86.125%	85.838%	87.413%
5	91.972%	91.972%	92.234%	433	78.707%	82.300%	81.892%	82.876%	81.103%	79.850%	83.530%	83.328%
6	88.108%	88.168%	88.643%	502	84.201%	90.060%	89.925%	91.411%	84.309%	83.956%	85.360%	86.335%
7	81.682%	81.682%	81.787%	611	86.216%	86.896%	87.003%	88.149%	81.117%	82.293%	83.564%	85.360%
8	69.114%	84.192%	84.181%	654	83.210%	85.061%	84.376%	86.035%	85.546%	86.374%	87.610%	87.718%
9	80.076%	80.076%	80.775%	610	91.511%	92.051%	92.259%	93.145%	91.159%	90.614%	91.900%	92.647%
10	77.013%	78.297%	78.925%	610	93.648%	92.856%	93.703%	94.123%	87.200%	86.058%	88.251%	87.986%
Ave	82.840%	84.699%	85.112%	542	83.103%	84.904%	84.976%	85.956%	84.480%	84.458%	85.957%	86.503%
Instance	s1c10b5				s1c10b10				s1c10b25			
	Initial	Control-Set	Local Search ILSSN	Beam Search f15 b5 stra5 Run Time (s)	Initial	Control-Set	Local Search ILSSN	Beam Search f15 b5 stra5 Run Time (s)	Initial	Control-Set	Local Search ILSSN	Beam Search f15 b5 stra5 Run Time (s)
1	82.172%	82.350%	82.866%	496	79.850%	80.597%	81.034%	81.480%	80.828%	84.075%	83.528%	85.762%
2	86.578%	86.791%	87.115%	421	81.804%	82.225%	82.530%	82.842%	89.162%	88.314%	89.797%	90.824%
3	75.661%	79.949%	79.835%	507	79.359%	79.359%	79.359%	79.582%	77.636%	78.139%	79.967%	81.099%
4	71.907%	72.225%	72.741%	596	75.925%	76.630%	76.722%	79.109%	81.152%	82.858%	83.564%	85.291%
5	76.208%	76.619%	76.920%	579	84.638%	86.173%	85.706%	86.781%	78.378%	78.592%	80.356%	80.967%
6	84.083%	84.227%	84.471%	492	88.433%	89.847%	90.172%	90.555%	79.364%	83.254%	82.538%	85.051%
7	87.833%	88.265%	89.168%	422	84.122%	88.806%	88.742%	89.487%	81.750%	82.840%	82.912%	84.205%
8	69.084%	76.127%	75.834%	587	72.705%	73.475%	73.933%	74.621%	76.577%	77.375%	78.223%	78.259%
9	75.637%	75.799%	77.193%	556	79.158%	80.865%	81.279%	82.146%	89.202%	88.351%	90.329%	90.852%
10	75.595%	76.382%	76.725%	580	84.619%	84.887%	84.652%	85.531%	84.306%	84.177%	85.625%	86.042%
Ave	78.476%	79.873%	80.287%	524	81.061%	82.286%	82.413%	83.213%	81.835%	82.798%	83.684%	84.835%

Table 5.15 soton1 Results Comparison among Initial Solution, Control-Set, ILS with Simple Neighbourhood, and Beam Search

Table 5.16 compares the results among initial solution, Control-Set, Iterated Local Search (ILS) and Beam Search (BS) for data sets soton2. We set beam search parameters as [f15 b5 stra10] for data sets s2c5b5, s2c5b10 and s2c5b25, and [f10 b5 stra5] for data sets s2c10b5, s2c10b10 and s2c10b25. For data set s2c5b5, beam search generates the best overall results except for instance 5, 7 and 10. ILS obtains the best results for instance 5 and 10, while Control-Set beats both algorithms on instance 7. The overall improvement rates over initial solutions are 0.513%, 0.515% and 0.780% for Control-Set, ILS and BS respectively. The average computation time for BS is 302 seconds. Similar outcomes are observed for the other five data sets. Beam search outperforms others on all averages (and for most instances), while ILS is able to show tiny edge on a few instances. We however want to present a few exceptions here. For data set s2c5b10, ‘Control-Set’ generates the same best result as BS on instance 3. For data set s2c5b25, beam search fails to reach the initial solution which is the best for instance 4. For data set s2c10b10 and s2c10b25, ‘Control-Set’ does better than ILS on average.

Instance	s2c5b5				s2c5b10				s2c5b25			
	Initial	Control-Set	Local Search ILSSN	Beam Search fl15 b5 stral0 Run Time (s)	Initial	Control-Set	Local Search ILSSN	Beam Search fl15 b5 stral0 Run Time (s)	Initial	Control-Set	Local Search ILSSN	Beam Search fl15 b5 stral0 Run Time (s)
1	93.478%	93.733%	94.017%	393	92.616%	92.887%	93.296%	93.225%	90.546%	92.313%	90.546%	92.725%
2	92.120%	92.761%	92.490%	283	94.878%	95.151%	95.017%	95.248%	93.810%	93.728%	93.810%	95.005%
3	89.831%	90.661%	90.374%	252	79.624%	80.460%	79.624%	80.460%	94.107%	93.196%	94.377%	94.600%
4	90.589%	91.371%	91.502%	190	94.088%	93.977%	94.375%	94.209%	96.139%	93.999%	96.139%	96.132%
5	91.390%	91.688%	91.996%	197	90.850%	93.141%	94.018%	93.755%	94.020%	93.892%	94.233%	94.504%
6	93.098%	94.235%	93.779%	213	94.484%	94.649%	94.944%	95.026%	94.438%	94.281%	94.438%	95.624%
7	95.296%	95.591%	95.417%	261	92.724%	93.204%	93.252%	93.640%	93.027%	92.680%	93.027%	93.942%
8	93.061%	93.747%	93.675%	262	94.831%	94.378%	95.632%	94.531%	91.237%	92.535%	91.342%	93.353%
9	88.437%	88.894%	88.650%	612	95.765%	95.819%	95.783%	95.876%	93.745%	94.487%	94.326%	94.827%
10	94.298%	93.642%	94.444%	356	93.138%	94.232%	94.207%	94.542%	92.953%	93.187%	93.755%	94.070%
Ave	92.160%	92.632%	92.634%	302	92.300%	92.790%	93.015%	93.051%	93.402%	93.430%	93.599%	94.478%
Instance	s2c10b5				s2c10b10				s2c10b25			
	Initial	Control-Set	Local Search ILSSN	Beam Search fl10 b5 stral5 Run Time (s)	Initial	Control-Set	Local Search ILSSN	Beam Search fl10 b5 stral5 Run Time (s)	Initial	Control-Set	Local Search ILSSN	Beam Search fl10 b5 stral5 Run Time (s)
1	95.331%	95.689%	95.365%	387	93.979%	94.465%	94.124%	94.949%	94.357%	94.044%	94.386%	94.687%
2	93.623%	94.075%	93.828%	358	93.940%	93.919%	94.021%	94.027%	93.615%	93.601%	93.803%	93.916%
3	93.908%	94.519%	94.667%	531	95.845%	96.199%	96.277%	96.482%	94.223%	94.069%	94.223%	94.388%
4	95.029%	94.723%	95.763%	349	95.783%	95.251%	96.279%	95.476%	93.421%	94.149%	93.763%	94.907%
5	94.613%	91.655%	94.653%	346	95.535%	95.964%	95.709%	96.151%	92.783%	93.652%	92.891%	94.014%
6	93.415%	93.805%	93.918%	366	94.476%	94.932%	94.768%	95.219%	93.865%	94.703%	93.865%	95.295%
7	90.693%	90.881%	91.150%	353	93.869%	94.393%	94.272%	94.541%	79.024%	80.405%	79.602%	81.543%
8	95.251%	96.096%	95.718%	498	91.346%	93.046%	92.167%	93.795%	91.922%	94.095%	91.922%	94.667%
9	91.372%	92.139%	91.884%	436	94.854%	96.341%	95.990%	96.567%	93.297%	94.129%	93.751%	94.856%
10	93.524%	95.462%	95.541%	318	96.470%	96.764%	97.031%	97.356%	94.011%	94.202%	94.011%	94.850%
Ave	93.676%	93.905%	94.249%	394	94.610%	95.127%	95.064%	95.456%	92.052%	92.705%	92.222%	93.312%

Table 5.16 soton2 Results Comparison among Initial Solution, Control-Set, ILS with Simple Neighbourhood, and Beam Search

Some conclusions can be drawn from these comparisons. Firstly, Control-Set tends to do very well when there are fewer box types. This is because for data set with 5 box types there are only 120 combinations of possible sequencing. Based on our observation, all these 120 combinations are run through by Control-Set during 500-second running time. That is to say the results generated by Control-Set for s1c5b5, s1c10b5, s2c5b5 and s2c10b5 are the upper-bound results without splitting box types. Since both ILS and BS have their results improved over those of Control-Set, we can assume that splitting box types is an implementation which has a positive impact on the final solution. For data sets with 25 box types, Control-Set performs badly in a couple of cases. It fails to improve on s1c5b25, and only manages to improve very slimly on s2c5b25. This is because it simply fails to form those better sequencings within 500 seconds. It manages to improve 1.176% for s1c10b25 but still falls short compared to ILS's 2.259% improvement rate. Data set s2c10b25 is an exception as Control-Set performs better than ILS with one of those lucky lotteries. Secondly, the results of ILS give mixed information. For some data sets, its results are quite mediocre. Its results only manage very slim improvement over those of Control-Set on data sets s1c5b10, s1c10b10 and s2c5b5. It fails to beat Control-Set on s2c10b10 and s2c10b25. It performs well on data set s1c5b25, s1c10b25 and s2c10b5, and generates decent results on the rest data sets. Despite extra coding effort, both Control-Set and ILS run for the same amount of time. Therefore at the consumer end, a better result with same running time is a still a gain towards them. Finally, beam search overall is the best performer. It manages to generate best average results for all data sets and best results for most instances. Also, its computation times are as competitive as those of Control-Set and ILS in most cases. It has similar computation times for data set s1c5b5, s1c5b10, s1c10b5, s1c10b10, s1c10b25 and s2c10b10. It uses much shorter computation times for data set s2c5b5 and s2c10b5. It uses more time for s1c5b25 and s2c5b10, and unavoidably needs much more time for s2c5b25 and s2c10b25 given the same parameter settings we set for data sets s2c5 and s2c10.

5.7 Conclusion

The main contribution of this section is that we design different iterated local search algorithms to improve the initial solution generated by Cargo Manager's constructive heuristic. We also design beam search algorithm which by nature works differently from iterated local search algorithms. The uniqueness of this work is that the constructive heuristic is presented as a standalone program, and the way on how its constructive heuristics exactly works remains uncontrollable to us. While iterated local search makes improvement based on rear-

ranging the original packings in the containers, the final improved solutions are kept feasible for industry use. The difficulty faced to solve this type of problem is obvious, as improvement heuristic is usually designed based on the known characteristics of the constructive heuristic. After the attempt to intervene how the constructive heuristic works in our iterated local search algorithms, we implement a completely different approach, beam search which builds partial solutions into a final one and leave constructive heuristic effectively do what it is good at without intervention. Results suggest that iterated local search algorithm with the design of straightforward objective-driven acceptance criteria and extensive neighbourhood search, in our case ILSSN, may generate better results over other more complicated ILS approaches, i.e. ILSIN and ILSDR. However, beam search remains the favor compared to ILS in general. We then compare the results of both approaches with that of Control-Set to draw a clue on if both approaches really do work. Beam search responds positively while iterated local search gives mixed signals.

Chapter 6

Heterogeneous Containers Problem

6.1 Problem Description

In this chapter, we would like to extend our beam search and iterated local search with simple neighborhood algorithms to solve heterogeneous containers problem. Homogeneous container problem is an input minimisation problem where the solution quality is evaluated through overall utilisation which is to be maximised. While heterogeneous containers problem still remains an input minimisation problem, the solution quality is evaluated by the total container cost used. The smaller the cost, the better the result.

The heterogeneous containers problem is defined as follows. A number of rectangular containers of M types is given. Containers with the same dimensions are categorised into the same container type, represented by C_1, C_2, \dots, C_M . Each container type is given a unique cost, represented by c_1, c_2, \dots, c_M . For unlimited heterogeneous containers problem, there are an unlimited number of each container available. For limited heterogeneous containers problem, there are m_t available containers of type t where $1 \leq t \leq M$. A given number of rectangular boxes are categorised into N types. All box types have different dimensions, and boxes of each type are all unique. Each box type is represented by B_1, B_2, \dots, B_N . There are n_j available boxes of type j where $1 \leq j \leq N$. Given a set of containers with different container types, the objective is to find the best combination of containers to allow all boxes to be orthogonally packed within the containers. The definition of the best will be that of minimising the total cost of containers used.

As we are looking for simulating a real industry situation, we will have only a few container types available. Thus we are solving the problem as Multiple Stock-Size Cutting Stock

Problem (MSSCSP), that is to pack weakly heterogeneous boxes into weakly heterogeneous containers. This aspect however does not reflect in our algorithm design, as our algorithms accommodate problems with both weakly and strongly heterogeneous containers. It rather represents the experiments we carry out in section 6.5. Same as for homogeneous container problem in the previous section, all solutions obtained here are approximate. On a separate note, the data sets we use in the experiments, i.e. *soton1* and *soton2*, are originally designed for homogeneous container problem with only the 40ft Standard Steel Maersk Container used during the data generation process.

The rest of the chapter is organised in such a way. Chapter 6.2 describes our revised beam search algorithm, and section 6.3 explains our revised iterated local search with simple neighbourhood algorithm. We modified beam search algorithm ahead of iterated local search because beam search appears to be the better approach to solve our real industry problem with Cargo Manager embedded as a standalone program, although we are aware that heterogeneous containers problem is not the same as homogeneous container problem in the obvious sense of different objective function with extra variables. Among all three iterated local search algorithms modeled in chapter 5, modification is only applied on iterated local search with simple neighbourhood because it is believed to be the most efficient among the three. The experiment design for heterogeneous containers problem is described in section 6.4 for both revised beam search and revised ILSSN algorithm. Complete results on this problem are then presented in section 6.5 with comparison on the results between these two algorithm presented in the end. Finally, a conclusion is drawn in section 6.6.

6.2 The Revised Beam Search Algorithm

In this section, we modify our beam search algorithm originally designed for homogeneous container problem. Modifications are designed for both unlimited and limited heterogeneous containers problem. The newly revised models are summarised in separate procedures.

6.2.1 Unlimited Heterogeneous Containers Problem

Compared to homogeneous container problem, three major modifications are made to adjust beam search algorithm to tackle the problem where unlimited heterogeneous containers are to be packed. These modifications happen within the stage of generating child nodes, local

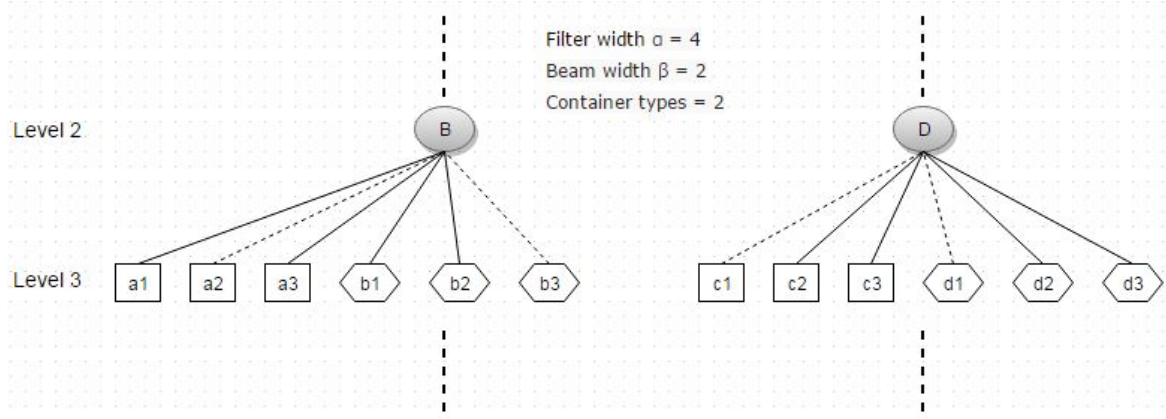


Fig. 6.1 Local Evaluation for Unlimited Heterogeneous Containers Problem

evaluation, and global evaluation.

Deriving from each beam node, child nodes are generated for all the container types. Different available strategies are applied on each container type. Take figure 6.1 as an example. Two container types are available. As beam width is 2, we have node B and D as beam nodes. Different strategies are applied to generated child nodes a1, a2 and a3 for the first container type from beam node B. Child nodes b1, b2 and b3 are generated using the same strategies for the second container type from beam node B. Beam node D has c1, c2 and c3 as child nodes for the first container type and d1, d2 and d3 for the second container type.

Given filter width α and N as the total number of container types, local evaluation compares utilisation among child nodes with same container type. The best $\frac{\alpha}{N}$ nodes are selected from each container type. Through such design, we ensure that all the container types are involved in the global evaluation stage as it is unfair to comparing nodes across different container types solely on utilisation. We believe that keeping all the container types in the global evaluation will result in a more competitive outcome of the next level beams. Take figure 6.1 as an example. We have filter width α as 4. That is 4 child nodes selected from beam node B, and 4 from D. Under beam node B, child node a1 and a3 are selected for the first container type, and b1 and b2 are selected for the second container type. Under the same reason, c2 and c3 along with d2 and d3 are selected from beam node D. In the case where filter width α can not be divided equally among N container types, the extra indivisible node(s) is randomly allocated among all container types and further selected based on its utilisation.

Given beam width β , the remaining unpacked boxes for each filter node are placed within

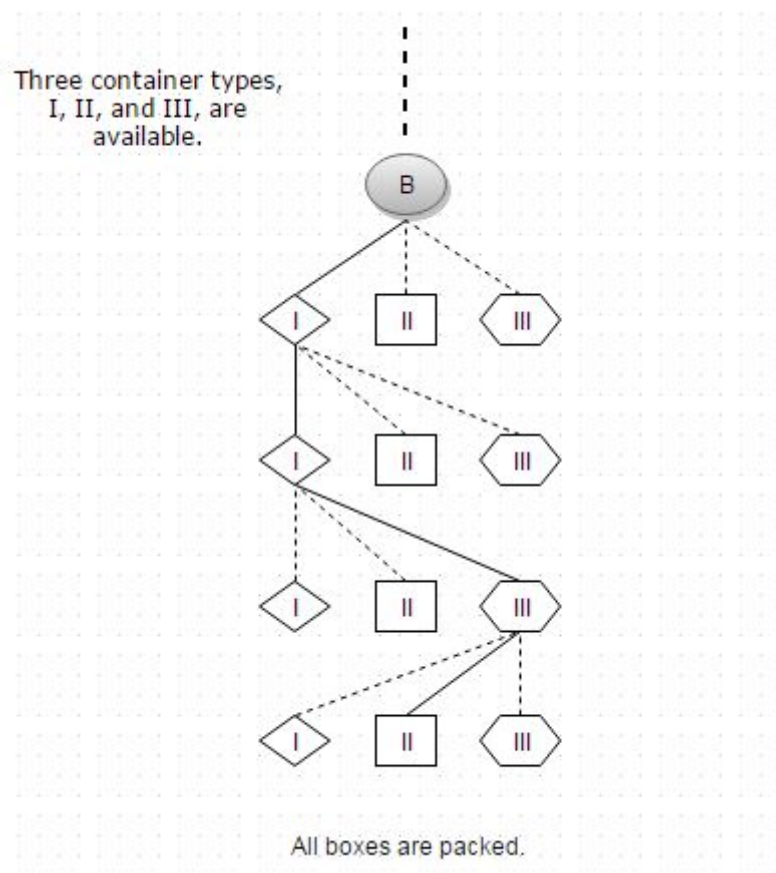


Fig. 6.2 Global Evaluation for Unlimited Heterogeneous Containers Problem

their original box types which are then sorted by default sequencing R . Each container type is used at each packing level, and the one with the best ratio of packed box volume and container cost is selected. This process ends when all boxes are packed. Global evaluation compares the total costs of the containers used for all filter nodes, and the best β nodes are selected as beam nodes. Take figure 6.2 as an example. There are three container types available. The remaining boxes for filter node B are packed using 4 containers, respectively as container type I, I, III and II. The decision of using each container is made by comparing the ratios of packed box volume and container cost among all three container types. For filter node B, The total cost is the costs of all packed containers up till B plus the costs of container I, I, III, and II.

Another modification worth of mentioning is the generation of the final solution and termination condition. Because of the existence of different container sizes, the final solution may be generated before the search reaches the end of the tree graph. When we generate child nodes using different strategies for each container type, one or more child nodes may be able to pack all the boxes. We define all the child nodes using the same container type from the same beam node as a cluster. If any child node within a cluster packs all the boxes, we compare the total cost of such node(s), or branch(es) just to be more accurate, with the best in record, and update it as the current best solution if it is better than the one in record. All the other nodes within the same cluster are pruned. The search terminates when there is no child node left for local evaluation. Also, a second termination condition is set before local evaluation to save computational time. For the remaining child nodes, we exclude any one from further evaluation if its current total cost plus the lowest container cost is larger or equal to the current best cost. The algorithm terminates when all nodes are excluded for further evaluation.

Our revised beam search algorithm for unlimited heterogeneous containers problem is summarised by the following procedure.

- N be the total number of container types
- C_P be the container type P where $P = I, II, III, \dots, N$
- V_P be the cost of container type P
- V_b be the best total cost of all the containers packed
- B be the total set of boxes
- b_r be the unpacked boxes

- T be the original box types
- c be the number of packed containers
- S_c be the partial solution containing c packed containers
- C_c be the first c containers packed
- V_c be the cost of the first c containers
- b_c be the boxes packed in the first c containers
- S_u be the partial solution for the current $(c + 1)th$ container
- C_u be the current container being packed that can be any container type C_P
- V_u be the cost of container C_u
- b_u be the number of boxes packed in the current $(c + 1)th$ container
- s_n be the proposed strategy n , $n = 0, 1, \dots, 6$
- α be filter width
- β be beam width
- $GLOBAL(S_c + S_u, b_c + b_u, B)$ be the global evaluation function that returns the total cost of all the containers used, where the packing order of the first $(b_c + b_u)$ boxes in the partial solution $(S_c + S_u)$ have been determined by the search tree through the choice of C_P and s_n in the first $(c + u)th$ containers, and the remaining $(B - b_c - b_u)$ unpacked boxes are placed and packed within their original box types T which are ordered using the fixed sequencing rule R
- $LOCAL(S_u, b_u)$ be the local evaluation function that returns the utilisation of container C_u

Step 0 Set $c = 0$, $S_0 = \emptyset$, $V_b = +\infty$ and root node as the single beam node. Initialize $b_r = B$.

Step 1 Node generation. Generate S_u (i.e. level $c + 1$ nodes) by adopting s_n , where $n = 0, 1, \dots, 6$, for each container type C_P .

Step 2 Update the best total cost. For each node that satisfies $b_u = b_r$, let $V_b = V_c + V_u$ if $V_c + V_u < V_b$. Disqualify each of such nodes as future filter node, and prune those nodes share the same beam and container type C_P .

- Step 3** Termination test 1. If all container types from all beams have one or more nodes that satisfies $b_u = b_r$, go to Step 8.
- Step 4** Termination test 2. Disqualify each node that satisfies $V_c + V_u + \text{the lowest } V_P \geq V_b$ from further evaluation. If no available node remains, go to Step 8.
- Step 5** Filtering. For each node compute $LOCAL(S_u, b_u)$. Compare nodes with same container type C_P and select the best $\frac{\alpha}{N}$ nodes from each container type. Prune remaining nodes. If the number of available nodes from the same beam and container type is less than $\frac{\alpha}{N}$, all nodes are selected as filtered nodes.
- Step 6** Select beam nodes. Compute $GLOBAL(S_c + S_u, b_c + b_u, B)$ for each of the remaining filter nodes. Select the best β nodes as beam nodes. If the number of filtered nodes is less than β , all filtered nodes are selected as beam nodes.
- Step 7** Update sets. For each selected beam node, form S_{c+1} by adding the beam node S_u to S_c , and remove b_u from b_r . Let $c = c + 1$, and go back to Step 1.
- Step 8** Final solution. Record the solution with V_b as the final solution.

6.2.2 Limited Heterogeneous Containers Problem

Compared to unlimited heterogeneous containers problem, limited heterogeneous containers problem has the extra condition that all container types have different available container numbers. Therefore, it is important to pay attention to the situation where not all container types are available. Take figure 6.3 as an example. When generating child nodes, all three container types are still available for beam node B but only two container types are available for beam D. For local evaluation, the filter width α is 6. Child nodes a1, a2, b1, b3, c1 and c2 are selected as filter nodes using the same method in unlimited heterogeneous containers problem. Because there are only two available container types available for beam node D, three filter nodes are to be selected for each container type. Therefore, all six child nodes are selected in this case. Also, not all containers may be available at each packing level during global evaluation. In figure 6.4 where the available number of container type I is only 1, once it is used it will not be available for the next packing. Assuming figure 6.2 and figure 6.4 has the same filter node B, the packing patterns are clearly different as for limited heterogeneous containers problem the four packed containers are container type I, III, II and II.

Another important modification is the feasibility check. After generating a child node, the

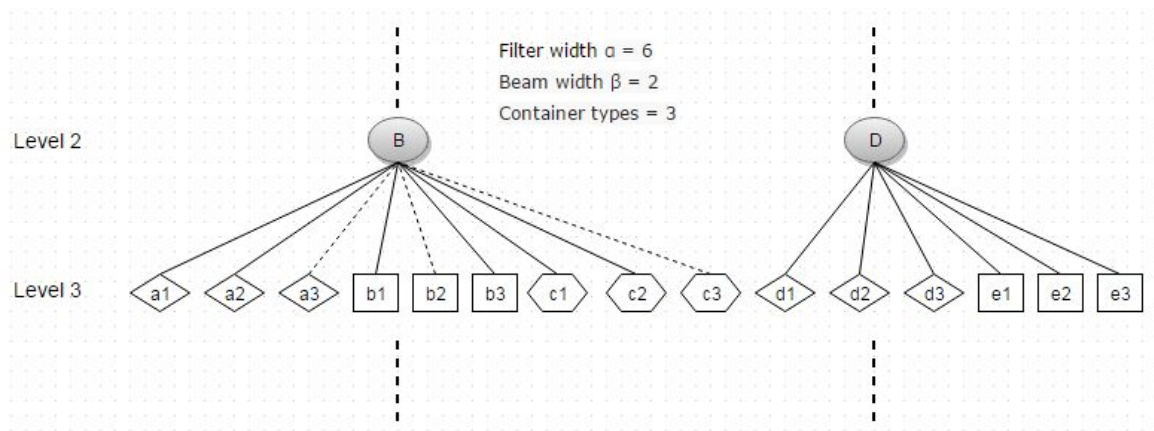


Fig. 6.3 Local Evaluation for Limited Heterogeneous Containers Problem

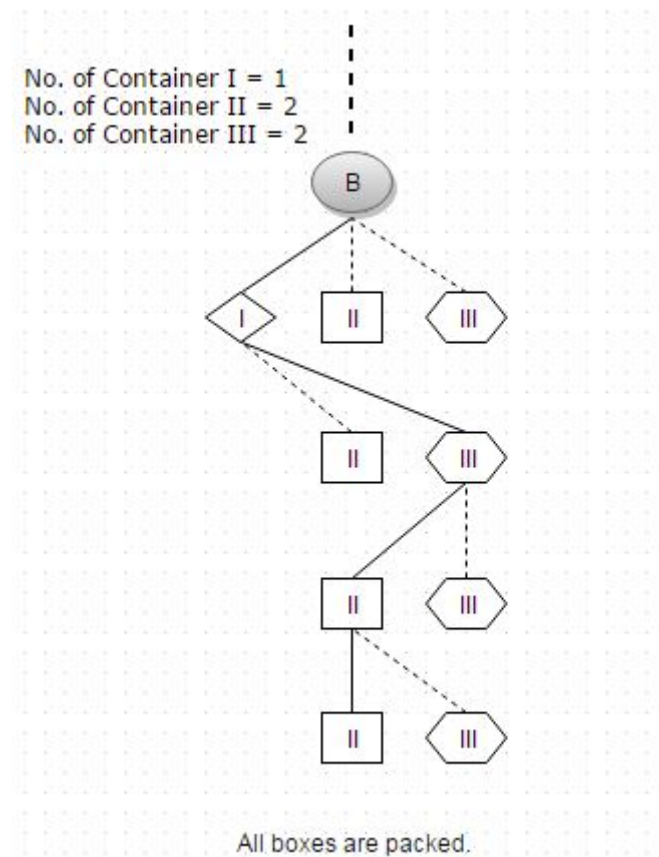


Fig. 6.4 Global Evaluation for Limited Heterogeneous Containers Problem

dimensions and volume of the remaining boxes and available containers are checked to ensure that there is container(s) larger enough available for the bigger boxes to fit in. At this stage, we only compare the volume of bigger boxes to that of larger containers without actual packing. Once container volume is larger and equal to the box volume, we assume the situation as feasible. This is to save computational time for further evaluation on obvious infeasible nodes without spending extra computational time on the checking process. If such a container does not exist or does not have enough space, the child node is declared as infeasible and is pruned immediately. The second feasibility check point is at the global evaluation where we actually pack the remaining boxes to obtain the total cost. The check point is set right after packing each available container type so the obvious infeasible packing is excluded in the comparison of different container types. At the end of packing each filter node, the node is pruned if not all boxes can be packed. It is worth to mention that strategy 0 automatically serves as the re-filter here if all the nodes generated under other strategies happen to be infeasible. Because strategy 0 places boxes within their original types and sorts them non-increasingly by box volume and height, bigger boxes will always be given priority to be packed first. Therefore, there should always be feasible nodes generated under strategy 0 when a feasible box-container combination is provided.

Compared to the procedure for unlimited heterogeneous containers problem in section 6.2.1, the procedure for limited heterogeneous containers problem slightly differs in a few places. A new notation, N_P , is defined as the number of containers for container type P . Since limited container problem introduces the number limit on each container type, there might be cases where one or more container type is not available. Therefore, during the global evaluation, Step 1, 3, 4 and 5 the availability of each container type should be checked. Moreover, for each selected beam node in Step 7 the container number of the selected container type is reduced by 1.

6.3 The Revised Iterated Local Search with Simple Neighbourhood (ILSSN)

In this section, we modify our Iterated Local Search with Simple Neighbourhood (ILSSN) algorithm originally designed for homogeneous container problem. Modifications are designed for both unlimited and limited heterogeneous containers problem. A newly revised procedure is summarised for unlimited heterogeneous containers problem. Since no new notation is needed, same notations in section 5.4.1 are used.

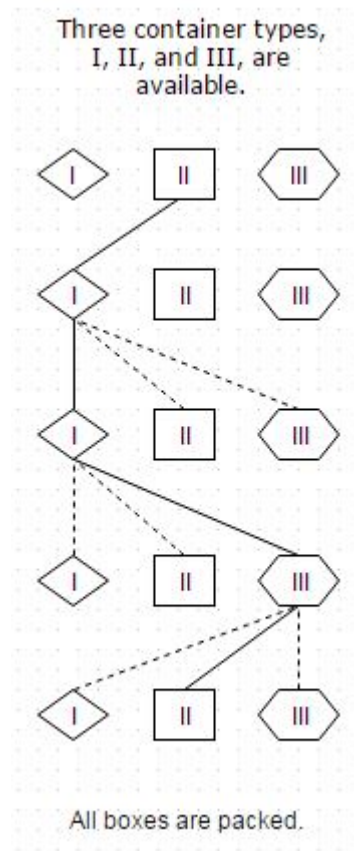


Fig. 6.5 Unlimited Heterogeneous ILSSN Initial Solution Generation

6.3.1 Unlimited Heterogeneous Containers Problem

We try to maintain the structure of the original Iterated Local Search with Simple Neighbourhood (ILSSN) while making a few modifications for solving unlimited heterogeneous containers problem. Major changes are made on the generation of initial solution, criterion on picking target container C_t , all the evaluation on the used container length including optional Intra Container Moves, the generation of complete solution before kick, and kick.

We include all container types when generating initial solution. All original batches defined as in section 4.3.1 are ordered under fixed sequencing rule R which is derived from our experiment in section 4.5.2 and will be revealed in section 6.4.3. All the boxes are packed by each container once. The container with the highest ratio of packed box volume and container cost is chosen. The packed boxes are removed from the next packing, and the same process continues till all the boxes are packed. Figure 6.5 demonstrates an example of initial solution generation. Three container types I, II and III are available. The first packed

container type is II, the second one is I, the third one is I, the fourth one is III, and the fifth and final one is II. Since each container has a unique cost attached to it, we then sum up the total cost as the initial objective value.

During Inter Container Optimisation, we choose target container C_t based on a different criterion than the utilisation we use in solving homogeneous container problem. Since the ratio of packed box volume and container cost is the criterion to judge how good a container packing in heterogeneous containers problem, we choose our target container C_t based on the ratio of packed box volume and container cost. Ratios of all packed containers are compared, and the one with the lowest ratio is chosen as the target container C_t .

Since the sum of whole used container cost is the objective function, the concept of used container length or the shortest length used among containers is no more valid here. Therefore, we remove all the evaluation on used container length. The container length used for target container does not have any effect on the objective function. As illustrated in algorithm 4 starting from step 15, a higher utilisation from the newly packed untarget container is the only indicator for an improvement. This is different from the algorithm for homogeneous container problem. In algorithm 3 because container length used in target container contributes to the objective function, even the newly packed untarget container has the same utilisation as before it is still considered as an improvement if the container length used in the target container is shortened. Same approach applies to the generation of the final solution. For homogeneous container problem, we find the container with the least used length and assumption is made that the unused part can be re-sold. Therefore, to generate the final solution is to find the least container length used among all packed containers. For the new unlimited heterogeneous containers problem presented in this section, used container length is not relevant any more because of the different container sizes. We take each container as a whole and present them with their costs.

Intra Container Optimisation, as presented in section 5.4.5, repacks each container with the boxes already packed within it. Improvement is made if the used container length is shortened for the same container. Inter Container Optimisation, as described in section 5.4.4, is then applied to all the improved containers. Although evaluating used container length is highly involved in Intra Container Optimisation, the mean of it is to seek possibility to further improve the utilisation of untarget containers. Therefore, Intra Container Optimisation still has its relevance here. Based on the results shown in section 5.6.2, the application yields slightly better results in some cases. Given it does not take up significantly extra time to run, we applied Intra Container Optimisation before both the kick and the generation of final solution.

Before each kick when either a local optimal is reached or no improvement could be found among untarget containers, we repack target container to generate a complete solution which is then compared with the current best solution. For homogeneous container problem, we simply pack all the unpacked boxes using the only container type. For unlimited heterogeneous containers problem, we try to pack all the unpacked boxes using the container type with the lowest cost possible. We firstly pack using the original target container type. Once all boxes are packed, we repack them using the container type with the lower cost. The process stops when the lowest used container cost is reached.

The design of the two kicks described in section 5.4.6 does not take heterogeneous containers into account. Therefore, a new kick is implement to suit our new problem. We name the kick as Pack Different Container (pDC). We unpacked target container and a randomly selected untarget container. We then pack these unpacked boxes using a container type which is different from target container and the chosen untarget container. Such design guarantees the kick to explore onto a different neighbourhood other than the current one. If not all the boxes can be packed, the remaining boxes are packed using all container types and the one with the best ratio of packed box volume and container cost is chosen. Kick pDC ends when all the boxes are packed.

6.3.2 Limited Heterogeneous Containers Problem

A few changes are made to adjust the algorithm towards the limited heterogeneous containers problem. These adjustments happen within the generation of the initial solution, the generation of complete solution before kick, and kick. Firstly during the generation of the initial solution, as shown in the figure 6.6 the availability of each container is checked before each packing. Three container types are available. The number of container type I is 1, container type II's is 3, and container type III has 2 containers. The first container packed is container type II, and the second container packed is type I. Therefore, when packing the third container container type I is not available anymore. The third packed container is type III, and the fourth is type II so as the fifth which leaves no box unpacked. Secondly when we repack target container using the container type with the lowest cost possible during the generation of a complete solution before each kick, we only use a container type if it is available. Last modification is within the kick pDC. If no different container type is available from the target container and the randomly chosen untarget container, we keep randomly selecting another untarget container till a container type other than target container and chosen untarget container is available. If not all the boxes can be packed by this different container,

Algorithm 4 The Revised Iterated Local Search with Simple Neighborhood (RILSSN)

```

1: Generate initial solution  $S_i$  under fixed sequencing rule  $R$  by comparing all containers
2: while time limit is not reached do
3:   for all container  $c \in C$  do while  $C > 1$  do
4:     Pick target container  $c_t$  with the lowest ratio of packed box volume and container
       cost (and unpack it)
5:     for each box type  $b \in B$  within  $c_t$  do while  $B \neq \emptyset$  do
6:       Choose  $n$  number of target box type  $b_t$  by its pre-decided priority, the num-
       ber  $n$  is decided under certain Rule
7:       for each remaining  $c \in C - c_t$  do
8:         Select an untarget container  $c_u$  (and its all boxes  $b_u$ ) in their pre-decided
           order
9:         for each priority  $b_t$  can be (while keeping relative priority among  $b_u$ ) do
10:          pack  $b_u + b_t$  and accept the First Improvement
11:          if all boxes can be packed ( $b'_u = b_u + b_t$ ) then
12:            Update solution and return to Step 6 providing the target con-
              tainer hasn't been emptied
13:          else
14:            Check utilisation rate in the untarget container  $c_u$ 
15:            if  $U < U'$  then
16:              Repack target container  $c_t$  with remaining boxes  $b_u + b_t - b'_u$ 
17:              if  $b_u + b_t - b'_u$  are all packed in  $c_t$  then
18:                Update solution and return to Step 6 providing the target
                  container hasn't been emptied
19:              end if
20:            end if
21:          end if
22:        end for
23:      end for
24:    end for
25:  end for
26:  Compare  $S_c$  with  $S_b$ 
27:  if  $S_c < S_b$  then
28:     $S_b = S_c$ 
29:  end if
30:  Apply KICK (i.e. pAC) to jump out of current local optimal
31: end while

```

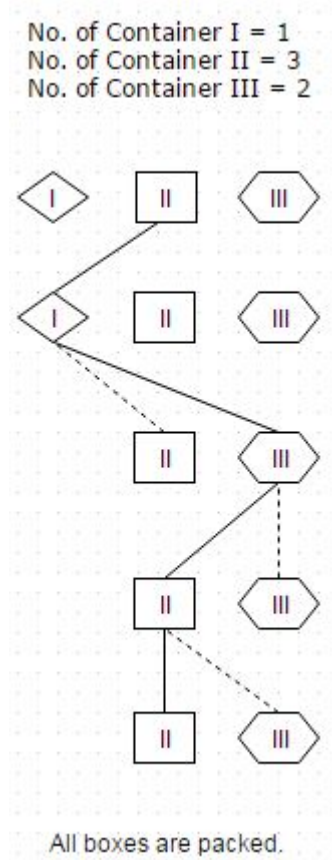


Fig. 6.6 Limited Heterogeneous ILSSN Initial Solution Generation

the remaining boxes are packed by the container with the best ratio of packed box volume and container cost among all available container types

In addition to the above three major changes, feasibility check similar to that of beam search (section 6.2.2) is implemented for limited heterogeneous containers problem. After finishing packing each container when generating the initial solution, we check the remaining boxes and available container types. If one or more remaining boxes does not fit into the available container types, we unpack the last packed container and prioritize those unfit boxes. Moreover, a box type is selected from target container during Inter Container Optimisation phase of the way same as in algorithm 3 and 4. When each untarget container is selected in turn to pack that target box type, the untarget container is skipped if the target box type does not fit into the untarget container. Also, after packed some or all target boxes the untarget container is skipped if any unpacked box type returned from untarget container does not fit target container. Moving on to the generation of a complete solution before kick, while we try to repack target container using the container type with the lowest cost

possible we obviously have to skip any container type which does not accommodate any remaining unpacked box type. Lastly, we randomly pick the container type same as either target container or the chosen untarget container if no different container type, which allows all unpacked box types to be fit in, is available.

6.4 Experimental Design

In this section, experimental design is explained for heterogeneous containers problem itself as well as for beam search (BS) and iterated local search with simple neighbourhood (ILSSN). In section 6.4.1, we will describe our choice of container types and the different cost settings we assigned to them. The reason behind our decision is also explained. We then set parameters as fixed values for BS and ILSSN.

6.4.1 Container Types, Costs and Container Numbers

As we try to simulate a real industry situation where only a few container types are commonly used, we select a combination of three different container types in our experiment. Table 6.1 summarises the complete container information of our design. The first container type is the 40ft Standard Steel Maersk Container with the size of 12035 * 2350 * 2393(mm). Then we choose the 20ft Std Steel Maersk Container with the size of 5896 * 2350 * 2385(mm). The last container we pick is the 10ft Standard Container, which is used by many shipping agents, with the size of 2800 * 2330 * 2350(mm). Three sets of costs are applied in our experiment. The first set of costs is called Reality Cost. These are the quotes provided by container rental agents. We compared a few quotes and adapt the most common ones. We set 6 as the value of 10ft container, 7 as the value of 20ft container, and 12 as the value of 40ft container. It is obvious that 40ft container has the advantage over 20ft and 10ft containers because of the cost attached to it. Under Reality Price 40ft container has a volume 4.4 times larger compared to 10ft container while its cost is only doubled compared to the later. It is not surprised that 40ft container will be preferred providing there are enough goods to be packed. One can almost expect that 20ft or 10ft container will only be considered as the last container when not enough goods are left. As a result of such prediction, we decide to make additional cost settings to help investigate the performances when the ‘fairer’ costs are set. We firstly set up a so-called Staged Cost. Under this setting, the cost of a container type is proportional to its volume. The purpose of this setting to give no advantage to any of the container type so they can compete with each other at any

Container Type	Reality Cost	Compromised Cost	Staged Cost
40ft Container (12035 * 2350 * 2393mm)	12	19	26.5
20ft Container (5896 * 2350 * 2385mm)	7	10	13
10ft Container (2800 * 2330 * 2350mm)	6	6	6

Table 6.1 Costs of Container Types under Different Settings

stage of the packing process. Therefore, the cost of 40ft container is set as 26.5. The cost of 20ft container is 13, and the cost of 10ft container is 6. Another set of costs, namely as Compromised Cost, sets the cost in the middle way of Reality Cost and Staged Cost. It still gives 40ft container cost advantage over the other two container types but not as dramatic as that in Reality Cost. The cost of 40ft container is 19, that of 20ft container is 10, and that of 10ft container is 6. For unlimited heterogeneous containers problem, the number of each container type is set to unlimited.

For limited heterogeneous containers problem, we make two different settings on container numbers. We create the first setting in such a way that all boxes can be technically packed using any two container types only. Therefore, we have 3 40ft containers, 6 20ft containers and 12 10ft containers for data set s1c5 and s2c5. For data set s1c10 and s2c10, numbers of all container types are doubled to 6 40ft containers, 12 20ft containers, and 24 10ft containers. We then create the other setting with the intention of using all three container types to pack all boxes. We have 2 40ft containers, 5 20ft containers and 12 10ft containers for data set s1c5 and s2c5, and 4 40ft containers, 10 20ft containers and 24 10ft containers for data set s1c10 and s2c10. All experiments are run under Reality Cost as we intend to simulate a real industry case.

6.4.2 Beam Search

Just like experiment on homogeneous container problem, different parameter settings, namely filter width, beam width and the number of child nodes generated under Strategy 1 to 6, are applied on different data sets for both unlimited and limited heterogeneous containers problem. All three cost settings are experimented on unlimited heterogeneous containers problem, and only Reality Cost is applied on limited heterogeneous containers problem. For unlimited heterogeneous containers problem, we set filter width = 15, beam width = 5 and the number of child nodes generated under Strategy 1 to 6 = 5 [f15 b5 stra5] for s1c5. Parameter setting for data set s1c10 is [f6 b3 stra4], [f9 b4 stra4] for s2c5, and [f6 b3 stra4] for s2c10. The aim is to let the running times of those data sets with 10 box types be as

close as 500 seconds under Reality Cost. The data sets with 5 box types are likely to finish running within 500 seconds, and those with 25 box types are expected to spend much more than 500 seconds. Also, those experiments under Staged Cost need more running times as more 10ft containers are expected to be used to result more levels in the beam search. For limited heterogeneous containers problem, parameter setting of [f15 b5 stra5] is applied on data sets s1c5 and s2c5 while [f9 b3 stra4] is set for data sets s1c10 and s2c10. The reason behind such setting is that it is hard to predict packing behavior of the algorithm when limited containers are provided. Also, we believe that adequate filter width, beam width and child nodes should be allowed in the tree structure to ensure the quality of the result. As the result of such setting, the experiment running time of Iterated Local Search will be based on the actual running time of beam search instead of the usual 500 seconds.

6.4.3 Iterated Local Search with Simple Neighbourhood

Section 5.6.2 showed that results under experiment allBoxes are generally better than those generated under other conditions for Iterated Local Search with Simple Neighbourhood (ILSSN). Hence, for Heterogeneous Containers Problem one of the conditions we run ILSSN under is all boxes when deciding the number of target boxes. The other condition is when selecting the target box type, we use non-increasing single box volume. Since we designed a new kick called pack Different Container (pDC), it remains as the only kick for Heterogeneous Containers Problem. As we discussed earlier in the revised ILSSN algorithm, we run both ICO before each kick and ICO when 500 seconds running time is reached. Therefore, the run uses the conditions of: non-increasing single box volume, all boxes, pack Different Container (pDC), and both ICO before each kick and ICO when 500 seconds running time is reached. Other than run conditions, fixed sequencing rule R is kept same as the sequencing combination VHN and H2 is kept as the only constructive heuristic during the whole experiment.

6.5 Computational Results

In this section we report the results generated for heterogeneous containers problem. Firstly, we generate results using our beam search algorithm for both unlimited and limited heterogeneous containers problem. Then, results using our iterated local search with simple neighbourhood (ILSSN) are generated for both unlimited and limited heterogeneous containers problem. Finally, we compare results from both algorithms with each other and against

control-set results. All results are generated by using the new benchmark data sets from section 4.4.2. For each data set, only the first 10 instances are used. Each instance is run 5 times and the average is taken as the result for that instance. We coded our algorithms in Java 1.8.0_05 and all the instances are run on an Intel(R) 2.60 GHz PC with a 4.00 GB RAM.

6.5.1 Beam Search Results

This section contains four tables. Table 6.2 and table 6.3 summarise the results for unlimited heterogeneous containers problem. Table 6.2 summarises the results for data set *soton1*, and table 6.3 summarises the results for data set *soton2*. In both tables, results run under three different costs, namely reality, compromised and staged cost, are presented for each instance. For each instance run under each cost, results are presented with the number of containers used, total cost used, the number of container type A (40ft container) used, the number of container type B (20ft container) used, and the number of container type C (10ft container). Table 6.4 and table 6.5 summarise the results for limited heterogeneous containers problem. Table 6.4 summarises the results for data sets which are generated for fitting into 5 containers, i.e. *s1c5* and *s2c5*. Results run under two different settings on container numbers, i.e. [2A 5B 12C] and [3A 6B 12C], are presented for each instance. Table 6.5 summarises the results for data sets which are generated for fitting into 10 containers, i.e. *s1c10* and *s2c10*. For each instance, results run under two different settings on container numbers, i.e. [4A 10B 24C] and [6A 12B 24C], are presented. All instances are run under reality cost for limited heterogeneous containers problem.

Table 6.2 summaries the results for unlimited heterogeneous containers problem on data set *soton1*. While results under all three costs are presented for each instance, we have no intention of comparing them against each other. Because same container type has completely different cost, it makes any comparison on total cost used irrelevant. However, we are interested to find out any pattern we may observe and draw insights on it. The major finding is that almost all packed containers are the largest container type (i.e. A) under reality cost while significant smaller containers especially container type C are used under staged price. All instances follow the same pattern, and here we take data set *s1c10b10* as an example. Under reality cost, an average number of 11.46 containers is used. Among them, there are 10.12 container type A, 0.88 container type B and 0.46 container type C. Smaller container types (i.e. B and C) are hardly used. Under compromised cost, an average number of 12.22 containers is used. 9.6 of them are container type A, 1.6 of them are container type B, and

1.02 of them are container type C. Container type A is still heavily used, while a slight raise in number of container type B and C is found. Under staged cost, an average number of 22.72 containers is used. Only 5.14 of them are container type A, 5.1 of them are container type B, and 12.48 of them are container type C.

As we explained in section 6.4.1, under reality cost container type A is given a huge advantage because of the cost attached to it. For 1 unit of cost in container type A, you are allocated 19.47% more space than that in container type B and 120.74% more space than that in container type C. Therefore, it is not surprising that almost all packed containers are type A. A much well packed (in terms of utilisation) container type B or C may be regarded as a worse choice compared to a rather sloppily packed container type A. Container type B and C only have their competitive edge when packing the last container. They are used if all the remaining boxes can be packed using one of them. As for staged cost, each 1 unit of cost yields the same amount of space in all three container types. No cost advantage is given in this case. The degree of fitness between packed boxes and given container is more crucial in deciding if such packing stays as a part of final solution. Therefore, similar to homogeneous container problem utilisation again becomes the major criterion when evaluating the quality of solution. However, it is worth point out that the number of container type C used in each instance highly depends on box types within that instance and the actual packing. From the results, we did not observe that one container type is favoured than others in terms of total packed container volume. As for compromised cost, 1 unit of cost brings 7.79% more space in container type A than that in container type B, and 39.4% more space than that in container type C. Container type A is still a favourable choice under compromised price while its cost edge is less compared to that under reality cost. The results in table 6.2 justify the cost setting.

Table 6.3 summaries the results for unlimited heterogeneous containers problem on data set soton2. The same pattern as in table 6.2 is observed again. Thus, the major finding from table 6.2 stands here. Furthermore, we found out that container type C is used more under staged cost when comparing to that in table 6.2. This is because boxes in data set soton2 are generally smaller than those in data set soton1, while the dimensions of container type C remain unchanged larger ratio between box size and container size likely results in more packing patterns with higher utilisation.

Table 6.4 summarises the results for data sets which are generated for fitting into 5 containers, i.e. s1c5 and s2c5, for limited heterogeneous containers problem. Results under two different settings on container numbers, i.e. [2A 5B 12C] and [3A 6B 12C], are presented for each instance. As explained in section 6.4.1, setting [2A 5B 12C] is created with the

	Reality Cost						Compromised Cost						Staged Cost					
s1c5b5	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	
1	5	60	5	0	0		5	95	5	0	0		7	131	4	1	2	
2	6	72	6	0	0		6.6	114.6	5.4	1.2	0		12.2	156.5	1.4	7.8	3	
3	5.4	60.4	4.6	0.4	0.4		6	92	4	1	1		8.8	123.1	2.2	3.6	3	
4	6	61	4	1	1		6	92	4	1	1		7	124.4	3.2	2.4	1.4	
5	5.6	60.6	4.4	0.6	0.6		6	92	4	1	1		7.2	124	3.6	1	2.6	
6	5.6	62	4.6	0.8	0.2		5.2	95.2	4.8	0.4	0		7.8	130.3	3.8	0.8	3.2	
7	6	66	5	0	1		6	101.8	5	0.2	0.8		10.4	139.3	3	2.2	5.2	
8	7	78	6	0	1		8.2	120.8	5.2	1	2		11.4	127.4	4.4	0.6	6.4	
9	6	67	5	1	0		6	105	5	1	0		17.6	141.4	1.2	1.6	14.8	
10	6	68	5.2	0.8	0		6.2	106.2	5	1	0.2		13.8	146.2	2	3.2	8.6	
Ave	5.86	65.5	4.98	0.46	0.42		6.12	101.46	4.74	0.78	0.6		10.32	134.36	2.88	2.42	5.02	
s1c5b10	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	
1	6	67	5	1	0		6	105	5	1	0		14.8	138.5	1.4	3	10.4	
2	5	60	5	0	0		6	92	4	1	1		7.6	123.7	3.4	1.2	3	
3	6	72	6	0	0		6	114	6	0	0		9.6	156.6	3.6	3.6	2.4	
4	6	72	6	0	0		7	111	5	1	1		17.4	149.9	1.4	2.4	13.6	
5	6	67	5	1	0		6	104.2	5	0.8	0.2		12.8	140.3	1.8	3.8	7.2	
6	5.4	60.6	4.6	0.6	0.2		5.8	94.2	4.2	1.2	0.4		10	127.6	2	3.8	4.2	
7	5	60	5	0	0		5	95	5	0	0		9.8	130.2	2.8	2	5	
8	6	66	5	0	1		6	101	5	0	1		10.8	135.2	2	4.2	4.6	
9	5.6	60.6	4.4	0.6	0.6		6	92	4	1	1		8.8	123	2.4	3	3.4	
10	6	61	4	1	1		6	92	4	1	1		10	122	2	3	5	
Ave	5.7	64.62	5	0.42	0.28		5.98	100.04	4.72	0.7	0.56		11.16	134.7	2.28	3	5.88	
s1c5b25	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	
1	6	67	5	1	0		6.8	104.6	4.6	1	1.2		15.2	138.2	1.2	3.2	10.8	
2	6	66	5	0	1		6	101	5	0	1		11.2	134.5	2.6	2	6.6	
3	5	60	5	0	0		5	95	5	0	0		12	127.3	1.4	3.8	6.8	
4	5.2	60.4	4.8	0.4	0		5	95	5	0	0		11.8	131.5	1.8	3.4	6.6	
5	6	66	5	0	1		6	101	5	0	1		10.4	137.9	3	2	5.4	
6	5.8	64.8	5	0	0.8		5.6	98.6	5	0	0.6		12.2	133.7	2.2	2.2	7.8	
7	6.2	66.4	4.8	0.4	1		6	101	5	0	1		11.4	134.6	2	3.6	5.8	
8	5	60	5	0	0		5	95	5	0	0		10.2	130	2.4	2.8	5	
9	6	61	4	1	1		6	92	4	1	1		9	122.5	3	1	5	
10	5	60	5	0	0		5.2	95.2	4.8	0.4	0		10	128.7	2.6	2.2	5.2	
Ave	5.62	63.16	4.86	0.28	0.48		5.66	97.84	4.84	0.24	0.58		11.34	131.89	2.22	2.62	6.5	
s1c10b5	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	
1	11	127	10	1	0		11	200	10	1	0		19	272.5	5	8	6	
2	10.4	120.8	9.6	0.8	0		10.8	190.8	9.2	1.6	0		30.2	263.4	3.6	1.2	25.4	
3	11.8	136	10.8	0.4	0.6		12	214	10.8	0.4	0.8		15.4	292.4	8.8	2.8	3.8	
4	14	156	11.8	1.2	1		16.2	240.8	10	3.4	2.8		24.6	325.7	5	10.8	8.8	
5	12	138	11	0	1		13	216	10	2	1		32	291.6	2.4	7.2	22.4	
6	12	130.2	9.4	1.8	0.8		13	199	8.2	3.6	1.2		16.2	271.9	6.2	6.8	3.2	
7	10.2	120.4	9.8	0.4	0		11.6	193.8	9	1.8	0.8		15.8	266.1	7.4	2.8	5.6	
8	12.2	140.2	11	1	0.2		12.4	221.4	11	1	0.4		16.4	303.9	9	3	4.4	
9	12	138.2	11	0.2	0.8		12.2	215.2	10.8	0.4	1		25.6	296.3	0.2	19.8	5.6	
10	12	139	11	1	0		12.2	215.2	10.8	0.4	1		19.4	298.9	7.4	4.4	7.6	
Ave	11.76	134.58	10.54	0.78	0.44		12.44	210.62	9.98	1.56	0.9		21.46	288.27	5.5	6.68	9.28	
s1c10b10	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	
1	11.4	132.4	10.6	0.4	0.4		12	206	10	1	1		24.2	280.1	4.6	5.8	13.8	
2	11.4	129.4	10	1	0.4		12.8	205	9	2.8	1		24.8	274.2	4	6.2	14.6	
3	13.6	143.2	9.6	4	0		15.6	225	9	3.6	3		23.2	300.6	4.8	9	9.4	
4	12	138	11	0	1		12.4	215.4	10.6	0.8	1		27.8	291.9	4.6	4.4	18.8	
5	10	120	10	0	0		10.4	190.4	9.6	0.8	0		14.6	264.6	7.2	4.2	3.2	
6	11	121	9	1	1		11	187	9	1	1		26.8	251.6	3.2	3.6	20	
7	10	120	10	0	0		11.2	188	8.8	1.6	0.8		17.4	255.2	6.4	2.8	8.2	
8	13	144.8	11	0.8	1.2		13	225	11	1	1		29.8	305.2	4.8	4	21	
9	12	133.6	10	1.6	0.4		13.2	209.8	9	3.4	0.8		22.2	283.3	5	6.8	10.4	
10	10.2	121.2	10	0	0.2		10.6	193.6	10	0	0.6		16.4	267.2	6.8	4.2	5.4	
Ave	11.46	130.36	10.12	0.88	0.46		12.22	204.52	9.6	1.6	1.02		22.72	277.39	5.14	5.1	12.48	
s1c10b25	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C		Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	
1	11.2	126.8	9.8	0.8	0.6		11.4	196.4	9.6	0.8	1		23.2	268.5	4.6	5	13.6	
2	11	120.2	8.8	1.4	0.8		10.6	184.6	9	1	0.6		24.4	254.2	2.8	7.2	14.4	
3	11	132	11	0	0		12.6	206.6	9.4	2.2	1		21.8	281.3	4.2	9.2	8.4	
4	11	126	10	0	1		12.6	198.6	8.6	2.8	1.2		18	268.7	6.2	4.8	7	
5	11	132	11	0	0		13.2	208.2	9	3	1.2		25.2	281.9	4.6	5.2	15.4	
6	11.4	128.8	10	0.4	1		11.6	200.8	9.6	1.6	0.4		20.4	270.6	6	3.6	10.8	
7	11	127	10	1	0		11	200	10	1	0		25	272.4	4.4	4.6	16	
8	12	138	11	0	1		12.2	215.2	10.8	0.4	1		24.4	292.7	4.2	8.6	11.6	
9	10.8	117.4	8.4	2.2	0.2		11.4	183.4	7.8	3.4	0.2		19.8	251.1	4.2	6.6	9	
10	11	126.2	10	0.2	0.8		11.8	196.8	9.2	1.6	1		19.2	265.3	5	6.8	7.4	
Ave	11.14	127.44	10	0.6	0.54		11.84	199.06	9.3	1.78	0.76		22.14	270.67	4.62	6.16	11.36	

Table 6.2 Beam Search results for Unlimited Heterogeneous Containers Problem on data set soton1

	Reality Cost					Compromised Cost					Staged Cost				
s2c5b5	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	5.4	60.4	4.6	0.4	0.4	6	92	4	1	1	12.8	119.2	2	0.2	10.6
2	5.6	60.6	4.4	0.6	0.6	6	92	4	1	1	9.6	122.1	2.6	1.6	5.4
3	6	61	4	1	1	6	92	4	1	1	15.4	124.3	0.6	2.8	12
4	6	61	4	1	1	6	92	4	1	1	18	122	0	2	16
5	6	61	4	1	1	6	92	4	1	1	17.2	122.4	0.8	0.4	16
6	5	60	5	0	0	6	92	4	1	1	11.6	120.7	1.4	3.2	7
7	5	55	4	1	0	5	86	4	1	0	8.6	119	2.4	2.6	3.6
8	6	61	4	1	1	6	92	4	1	1	11.4	120.6	2	1.6	7.8
9	5	60	5	0	0	5.8	95.8	4.2	1.6	0	13.2	127.3	1.8	1.6	9.8
10	6	61	4	1	1	6	92	4	1	1	20	120	0	0	20
Ave	5.6	60.1	4.3	0.7	0.6	5.88	91.78	4.02	1.06	0.8	13.78	121.76	1.36	1.6	10.82
s2c5b10	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	6	61	4	1	1	6	92	4	1	1	16.4	122	0.4	2.2	13.8
2	5	55	4	1	1	5	86	4	1	0	14.4	118.1	1	1.6	11.8
3	6	70	5.6	0.4	0	6	105	5	1	0	9.2	145.5	4.2	0.6	4.4
4	5.4	60.4	4.6	0.4	0.4	6	92	4	1	1	12	120	2	1	9
5	6	61	4	1	1	6	92	4	1	1	10.6	121.3	2.2	1.8	6.6
6	5.8	60.8	4.2	0.8	0.8	5.6	89.6	4	1	0.6	12	120.1	1.8	1.6	8.6
7	6	61	4	1	1	6	92	4	1	1	10.8	121.2	2	2.2	6.6
8	5	60	5	0	0	6	92	4	1	1	13	119.4	1.2	2.4	9.4
9	5	55	4	1	0	5	86	4	1	0	15	117.5	1	1	13
10	5.4	60.4	4.6	0.4	0.4	6	92	4	1	1	12.2	120.1	1.4	2.6	8.2
Ave	5.56	60.46	4.4	0.7	0.46	5.76	91.86	4.1	1	0.66	12.56	122.52	1.72	1.7	9.14
s2c5b25	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	5.8	60.8	4.2	0.8	0.8	6	92	4	1	1	12.4	124.2	1.2	3.6	7.6
2	5	59	4.8	0.2	0	5.8	90.8	4	1	0.8	12.8	119.4	1.6	1.4	9.8
3	5.2	60.2	4.8	0.2	0.2	6	92	4	1	1	12.4	119.6	2	0.6	9.8
4	5	55	4	1	0	5	86	4	1	0	12.6	118.5	1	3.2	8.4
5	5	60	5	0	0	6	92	4	1	1	12.6	119.7	1.4	2.2	9
6	5	55	4	1	0	5	86	4	1	0	14	118.5	1	2	11
7	6	61	4	1	1	6	92	4	1	1	12	120.4	1.2	3.4	7.4
8	6	61	4	1	1	6.2	92.2	3.8	1.4	1	14	122.5	1.4	1.4	11.2
9	5	60	5	0	0	6	92	4	1	1	12.8	119.6	1.2	2.6	9
10	6	61	4	1	1	6	92	4	1	1	10.2	121.6	2.4	1.6	6.2
Ave	5.4	59.3	4.38	0.62	0.4	5.8	90.7	3.98	1.04	0.78	12.58	120.4	1.44	2.2	8.94
s2c10b5	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	9	108	9	0	0	10	172	8	2	0	30.8	234.9	0.6	5.4	24.8
2	10	114	9	0	1	10	177	9	0	1	16	240.2	5.6	4.2	6.2
3	9	108	9	0	0	9	171	9	0	0	18	238.6	4.8	4.6	8.6
4	9	108	9	0	0	9	171	9	0	0	20	236.5	5	2	13
5	10	115	9	1	0	10.4	181.4	8.6	1.8	0	27.2	247.4	2.4	5	19.8
6	9.4	110.4	9	0	0.4	9.4	173.4	9	0	0.4	17.6	238.8	5.2	3.8	8.6
7	10	115	9	1	0	10	181	9	1	0	17.6	248.2	6	2.8	8.8
8	9	108	9	0	0	9.6	171.6	8.4	1.2	0	16.4	235.5	5.8	2.6	8
9	10.4	115.8	8.6	1.8	0	10	180.2	9	0.8	0.2	23	244.5	2.6	7.6	12.8
10	9	108	9	0	0	9	171	9	0	0	25.8	237.2	3.2	2.4	20.2
Ave	9.48	111.02	8.96	0.38	0.14	9.64	174.96	8.8	0.68	0.16	21.24	240.18	4.12	4.04	13.08
s2c10b10	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	9.2	109.2	9	0	0.2	9	171	9	0	0	28.2	237.9	2.6	2.2	23.4
2	10	114	9	0	1	10	177	9	0	1	25.6	238.7	3.4	2.2	20
3	9	108	9	0	0	10	172	8	2	0	15.2	234.2	5.2	5.2	4.8
4	9	108	9	0	0	9.2	171.2	8.8	0.4	0	23.2	236.5	4.2	1.6	17.4
5	9	108	9	0	0	10	172	8	2	0	14.6	235.7	6.2	3	5.4
6	9	108	9	0	0	9	171	9	0	0	19.6	237.2	4.4	4.2	11
7	10	114	9	0	1	10	177	9	0	1	21.2	238.9	3.4	6	11.8
8	10	114	9	0	1	10	177	9	0	1	21.8	243.6	4	4.4	13.4
9	9	108	9	0	0	10.2	172.2	7.8	2.4	0	16.8	235.1	5.8	2.2	8.8
10	10	110	8	2	0	10.6	172.6	7.4	3.2	0	15.8	233.5	5.4	4	6.4
Ave	9.42	110.12	8.9	0.2	0.32	9.8	173.3	8.5	1	0.3	20.2	237.13	4.46	3.5	12.24
s2c10b25	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	10	114	9	0	1	10	177	9	0	1	17.2	239.3	5	4.8	7.4
2	10	114	9	0	1	10	177	9	0	1	22.8	241.3	3.8	3.8	15.2
3	10	114	9	0	1	10	177	9	0	1	19	240.2	5.2	2.8	11
4	10	114	9	0	1	9.8	175.8	9	0	0.8	18	238.4	5.2	3.4	9.4
5	10	114	9	0	1	10	177	9	0	1	20.8	241.6	4.4	3.8	12.6
6	9.2	108.4	8.8	0.4	0	9	171	9	0	0	19.2	237.6	4.4	4.6	10.2
7	11.8	133	10.2	1	0.6	12	206	10	1	1	28.4	281.5	4.6	2.4	21.4
8	10	113.2	8.8	0.4	0.8	9.8	175.8	9	0	0.8	19.8	240	4	5.6	10.2
9	9.2	109.2	9	0	0.2	9.2	172.2	9	0	0.2	24	238.9	3.4	3.6	17
10	9.8	112.8	9	0	0.8	9.8	175.8	9	0	0.8	17.4	239	5.2	4	8.2
Ave	10	114.66	9.08	0.18	0.74	9.96	178.46	9.1	0.1	0.76	20.66	243.78	4.52	3.88	12.26

Table 6.3 Beam Search results for Unlimited Heterogeneous Containers Problem on data set soton2

intention of packing all boxes using all three container types while all boxes can be packed using any two container types under setting [3A 6B 12C]. All experiments are run under reality cost. A couple of patterns are observed here. Firstly, total cost used under setting [3A 6B 12C] is always lower than that under setting [2A 5B 12C]. The reason is that more options are available when there are more containers available within each container type and in total. With more options to choose from, total cost used is likely to be driven down. The second finding further proves our point on the advantage of container type A under reality cost. Under setting [2A 5B 12C] where all three container types are needed to pack all boxes, all containers from container type A and B are used except for instance 8 from data set s1c5b5. Container type C is barely used while all containers from type A are packed under setting [3A 6B 12C] where all boxes can be packed using only container type A and B.

Table 6.5 summarises the results for data sets which are generated for fitting into 10 containers, i.e. s1c10 and s2c10, for limited heterogeneous containers problem. Results under two different settings on container numbers, i.e. [4A 10B 24C] and [6A 12B 24C], are presented for each instance. As explained in section 6.4.1, setting [4A 10B 24C] is created with the intention of packing all boxes using all three box types while all boxes can be packed using any two container types under setting [6A 12B 24C]. All experiments are run under reality cost. From the results, we again observe the same pattern which happens in table 6.4.

In summary, 40ft container type (A) has a significant cost advantage over 20ft container type (B) and 10ft container type (C) under reality cost. The utilisation of containers may not have as much impact as for solving homogeneous container problem on the final solution. On the other hand, staged cost resembles the homogeneous container problem as the container cost is proportional to the container volume. Smaller containers such as container type B and C therefore gain their ground to fairly compete with container type A, and are used more often compared to when they are under reality cost. We also list the average computational time of beam search for each data. Table 6.6 presents the running times under three different costs for unlimited heterogeneous containers problem. Table 6.7 presents the running times under different container-number settings for limited heterogeneous containers problem. In the next section, experiments on iterated local search will run the exact same amount of time as in here so that the final comparison between the two will be the most fair.

	2A 5B 12C					3A 6B 12C				
s1c5b5	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	10	77	2	5	3	8	70	3	4	1
2	14	101	2	5	7	9	78	3	6	0
3	8	65	2	5	1	7	63	3	3	1
4	9	71	2	5	2	7	63	3	3	1
5	9	71	2	5	2	7	63	3	3	1
6	10	77	2	5	3	7	64	3	4	0
7	11	83	2	5	4	8	71	3	5	0
8	16.2	114	2	4.8	9.4	10.8	87.4	3	4.6	3.2
9	11	83	2	5	4	8	71	3	5	0
10	13	95	2	5	6	9	77	3	5	1
Ave	11.12	83.7	2	4.98	4.14	8.08	70.74	3	4.26	0.82
s1c5b10	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	11	83	2	5	4	8	71	3	5	0
2	8	65	2	5	1	7	63	3	3	1
3	14.6	104.6	2	5	7.6	9.4	80.4	3	6	0.4
4	13	95	2	5	6	9	77.4	3	5.4	0.6
5	11	83	2	5	4	8	71	3	5	0
6	9	71	2	5	2	7	64	3	4	0
7	10	77	2	5	3	7	64	3	4	0
8	10	77	2	5	3	8	70	3	4	1
9	8	65	2	5	1	7	63	3	3	1
10	8	65	2	5	1	7	63	3	3	1
Ave	10.26	78.56	2	5	3.26	7.74	68.68	3	4.24	0.5
s1c5b25	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	11	83	2	5	4	8	71	3	5	0
2	10.6	80.6	2	5	3.6	8	70	3	4	1
3	9	71	2	5	2	7	64	3	4	0
4	10	77	2	5	3	7	64	3	4	0
5	11	83	2	5	4	8	70.8	3	4.8	0.2
6	10	77	2	5	3	8	70	3	4	1
7	11	83	2	5	4	8	70	3	4	1
8	10	77	2	5	3	7	64	3	4	0
9	8	65	2	5	1	7	63	3	3	1
10	9.6	74.6	2	5	2.6	7	64	3	4	0
Ave	10.02	77.12	2	5	3.02	7.5	67.08	3	4.08	0.42
s2c5b5	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	8	65	2	5	1	7	63	3	3	1
2	8	65	2	5	1	7	63	3	3	1
3	9	71	2	5	2	7	63.4	3	3.4	0.6
4	8	65	2	5	1	7	63	3	3	1
5	8	65	2	5	1	7	63	3	3	1
6	8	65	2	5	1	7	63	3	3	1
7	8	65	2	5	1	6	57	3	3	0
8	8	65	2	5	1	7	63	3	3	1
9	9	71	2	5	2	7	64	3	4	0
10	8	65	2	5	1	7	63	3	3	1
Ave	8.2	66.2	2	5	1.2	6.9	62.54	3	3.14	0.76
s2c5b10	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	8	65	2	5	1	7	63	3	3	1
2	8	65	2	5	1	7	63	3	3	1
3	12	89	2	5	5	8.6	74.4	3	4.8	0.8
4	8	65	2	5	1	7	63	3	3	1
5	8	65	2	5	1	7	63	3	3	1
6	8	65	2	5	1	7	63	3	3	1
7	8	65	2	5	1	7	63	3	3	1
8	8	65	2	5	1	7	63	3	3	1
9	7.4	61.4	2	5	0.4	6	57	3	3	0
10	8	65	2	5	1	7	63	3	3	1
Ave	8.34	67.04	2	5	1.34	7.06	63.54	3	3.18	0.88
s2c5b25	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	8	65	2	5	1	7	63	3	3	1
2	8	65	2	5	1	7	63	3	3	1
3	8	65	2	5	1	7	63	3	3	1
4	7	59	2	5	0	6	57	3	3	0
5	8	65	2	5	1	7	63	3	3	1
6	7.8	63.8	2	5	0.8	6	57	3	3	0
7	8	65	2	5	1	7	63	3	3	1
8	8	65	2	5	1	7	63	3	3	1
9	8	65	2	5	1	7	63	3	3	1
10	8	65	2	5	1	7	63	3	3	1
Ave	7.88	64.28	2	5	0.88	6.8	61.8	3	3	0.8

Table 6.4 Beam Search results for Limited Heterogeneous Containers Problem on data sets s1c5 and s2c5

	4A 10B 24C					6A 12B 24C				
s1c10b5	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	20.8	158.8	4	10	6.8	15	135	6	9	0
2	19	148	4	10	5	14.4	130.4	6	8	0.4
3	24.6	181.6	4	10	10.6	16.8	146.8	6	10	0.8
4	28.2	203.2	4	10	14.2	18.8	160.8	6	12	0.8
5	24	178	4	10	10	17	148.4	6	10.4	0.6
6	20.8	158.8	4	10	6.8	16.2	140.8	6	7.6	2.6
7	21.4	162.2	4	9.8	7.6	15.2	135	6	7.8	1.4
8	26.4	192.4	4	10	12.4	17.8	153.8	6	11	0.8
9	25	184	4	10	11	17	148.8	6	10.8	0.2
10	25.8	188.8	4	10	11.8	17.2	150	6	10.8	0.4
Ave	23.6	175.58	4	9.98	9.62	16.54	144.98	6	9.74	0.8
s1c10b10	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	22.2	167.2	4	10	8.2	16	141.8	6	9.8	0.2
2	21.2	161.2	4	10	7.2	16	141	6	9	1
3	27	196	4	10	13	18	154.2	6	10.2	1.8
4	25.2	185.2	4	10	11.2	17	149	6	11	0
5	19.6	151.6	4	10	5.6	14.8	132.8	6	8	0.8
6	17.4	138.4	4	10	3.4	14	127	6	7	1
7	18.4	144.4	4	10	4.4	14	127.8	6	7.8	0.2
8	26.8	194.8	4	10	12.8	18	155.8	6	11.8	0.2
9	22.2	167.2	4	10	8.2	16.2	143	6	9.8	0.4
10	20	154	4	10	6	15	134	6	8	1
Ave	22	166	4	10	8	15.9	140.64	6	9.24	0.66
s1c10b25	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	20	154	4	10	6	15	134.2	6	8.2	0.8
2	18	142	4	10	4	14	127	6	7	1
3	22	166	4	10	8	16	141.4	6	9.4	0.6
4	20	154	4	10	6	15	134.2	6	8.2	0.8
5	22.4	168.4	4	10	8.4	16.2	142.8	6	9.6	0.6
6	20.4	156.4	4	10	6.4	15.4	137	6	8.6	0.8
7	21.6	163.6	4	10	7.6	15	135	6	9	0
8	24.2	179.2	4	10	10.2	17	148.2	6	10.2	0.8
9	17.4	138.4	4	10	3.4	13.8	125.8	6	7	0.8
10	20.2	155.2	4	10	6.2	15.4	136.2	6	7.8	1.6
Ave	20.62	157.72	4	10	6.62	15.28	136.18	6	8.5	0.78
s2c10b5	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	14	118	4	10	0	12	114	6	6	0
2	15	124	4	10	1	13	120	6	6	1
3	15	124	4	10	1	13	120	6	6	1
4	15	124	4	10	1	12.8	118.8	6	6	0.8
5	16.4	132.4	4	10	2.4	13	121	6	7	0
6	15	124	4	10	1	13	120	6	6	1
7	16.2	131.2	4	10	2.2	13	121	6	7	0
8	14	118	4	10	0	12	114	6	6	0
9	16	130	4	10	2	13	121	6	7	0
10	15	124	4	10	1	12.2	115.2	6	6	0.2
Ave	15.16	124.96	4	10	1.16	12.7	118.5	6	6.3	0.4
s2c10b10	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	15	124	4	10	1	13	120	6	6	1
2	15	124	4	10	1	13	120	6	6	1
3	14	118	4	10	0	12	114	6	6	0
4	15	124	4	10	1	12.4	116.4	6	6	0.4
5	14	118	4	10	0	12	114	6	6	0
6	15	124	4	10	1	13	120	6	6	1
7	15	124	4	10	1	13	120	6	6	1
8	16	130	4	10	2	13	120.6	6	6.6	0.4
9	15	124	4	10	1	12	114	6	6	0
10	14	118	4	10	0	12	114	6	6	0
Ave	14.8	122.8	4	10	0.8	12.54	117.3	6	6.06	0.48
s2c10b25	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C	Cont No.	Cost	No. of CT A	No. of CT B	No. of CT C
1	15	124	4	10	1	13	120	6	6	1
2	15.6	127.6	4	10	1.6	13	120	6	6	1
3	15	124	4	10	1	13	120	6	6	1
4	15	124	4	10	1	13	120	6	6	1
5	15.6	127.6	4	10	1.6	13	120	6	6	1
6	15	124	4	10	1	13	120	6	6	1
7	22	166	4	10	8	16.2	143	6	9.8	0.4
8	15.4	126.4	4	10	1.4	13	120	6	6	1
9	15	124	4	10	1	13	120	6	6	1
10	15	124	4	10	1	13	120	6	6	1
Ave	15.86	129.16	4	10	1.86	13.32	122.3	6	6.38	0.94

Table 6.5 Beam Search results for Limited Heterogeneous Containers Problem on data sets s1c10 and s2c10

	Reality Cost	Compromised Cost	Staged Cost
s1c5b5	552	665	1529
s1c5b10	538	663	1687
s1c5b25	672	715	1913
s1c10b5	583	674	1623
s1c10b10	574	660	1717
s1c10b25	588	685	1617
s2c5b5	328	357	1364
s2c5b10	476	477	1564
s2c5b25	3235	3285	11429
s2c10b5	475	542	1762
s2c10b10	511	552	1780
s2c10b25	1435	1432	4584

Table 6.6 Beam Search Running Time (in seconds) for Unlimited Heterogeneous Containers Problem

	2A 5B 12C	4A 10B 24C	3A 6B 12C	6A 12B 24C
s1c5b5	854		659	
s1c5b10	760		618	
s1c5b25	955		799	
s1c10b5		1277		991
s1c10b10		1188		974
s1c10b25		1107		990
s2c5b5	541		520	
s2c5b10	713		682	
s2c5b25	3034		3516	
s2c10b5		762		658
s2c10b10		756		677
s2c10b25		1592		1459

Table 6.7 Beam Search Running Time (in seconds) for Limited Heterogeneous Containers Problem

6.5.2 Iterated Local Search with Simple Neighbourhood Results

Four tables are presented in this section. For both unlimited and limited heterogeneous containers problem, each data set is run for the exact same amount of time as those in table 6.6 and table 6.7. Table 6.8 and table 6.9 summarise the results for unlimited heterogeneous containers problem. Table 6.8 summarises the results for data set soton1, and table 6.9 summarises the results for data set soton2. In both tables, three different costs, namely reality, compromised and staged cost, are used to generate results for each instance. For each instance run under each cost, results are presented with the initial container number, best container number, initial cost used and best cost used. Table 6.10 and table 6.11 summarise the results for limited heterogeneous containers problem. Table 6.10 summarises the results for data sets which are generated for fitting into 5 containers, i.e. s1c5 and s2c5. Two different settings on container numbers, [2A 5B 12C] and [3A 6B 12C], are used to generate results for each instance. Table 6.11 summarises the results for data sets which are generated for fitting into 10 containers, i.e. s1c10 and s2c10. Results run under two different settings on container numbers, [4A 10B 24C] and [6A 12B 24C], are presented for each instance. All instances are run under reality cost for limited heterogeneous containers problem.

Table 6.8 summaries the results for unlimited heterogeneous containers problem on data set soton1. Just as beam search results in the previous chapter, here for each instance we do not compare results under different costs as such comparison would be irrelevant. Two patterns are observed across the whole table. Firstly, the majority of the instances from all data sets had improvement on the objective value, total cost, under all three cost settings. Secondly, with the improvement on total cost the number of containers is reduced under reality cost while more containers are used under staged cost. This is because 40ft container (type A) has an advantage on cost and therefore is preferred under reality price. The improvement is largely achieved through emptying a 10ft container (i.e. type C) into the rest larger containers, mainly container type A. Hence, container number is reduced under reality cost. On the other hand, cost on each container type fairly represents the volume of the container type under staged cost. Smaller container types B and C are as competitive as large container type A. Therefore, by packing boxes in the large container into a few smaller containers which have their overall volume below that of a larger container the total cost is reduced while the number of container used increases.

Table 6.9 summaries the results for unlimited heterogeneous containers problem on data set soton2. Compare to table 6.8, less improvement is observed for data set soton2. This is especially the case for results under reality and compromised cost, where not many improvements are observed. The reason behind this is because utilisation is not the major

impact factor when container cost does not represent container volume, improvement on utilisation may not be reflect on the total cost. The other reason is that boxes in soton2 are smaller boxes which are usually already tightly packed. Such outcome has been observed throughout the whole experiment starting on the homogeneous container problem and is well expected. In short, ILSSN algorithm is able to make improvement on total cost used for unlimited heterogeneous containers problem.

Table 6.10 summarises the results for data sets which are generated for fitting into 5 containers, i.e. s1c5 and s2c5, for limited heterogeneous containers problem. For each instance, two different settings on container numbers, [2A 5B 12C] and [3A 6B 12C], are used to generate results. Under setting [2A 5B 12C] all three container types are expected to be used in order to pack all boxes, while it is possible to use only two container types to pack all boxes under setting [3A 6B 12C]. All experiments are run under reality cost. Two observations are obtained here. Firstly, total cost under setting [3A 6B 12C] is lower than that under setting [2A 5B 12C] for every instance. This proves that under the same cost setting, total cost can be driven down when there are more containers in each container type to be chosen from. Secondly, decent improvement on total cost is observed on data set s1c5 while improvement on s2c5 is minor. Similar patterns are observed for data sets s1c10 and s2c10 which have their results summarised in table 6.11. Compared to setting [4A 10B 24C], more options on container selection under setting [6A 12B 24C] lead to lower total cost for every instance.

6.5.3 Control-Set and Results Comparison

In this session, we compare the results of beam search (BS) with those of iterated local search with simple neighbourhood (ILSSN) for each setting. There are overall seven settings, and the results comparison under each setting is presented in a separate table. Three settings are cost setting for unlimited heterogeneous containers problem, i.e. reality (table 6.14), compromised (table 6.13) and staged (table 6.12) cost. For limited heterogeneous containers problem, settings [2A 5B 12C] (table 6.15) and [3A 6B 12C] (table 6.16) on container numbers are designed for data sets s1c5 and s2c5 while the last two settings on container numbers, [4A 10B 24C] (table 6.17) and [6A 12B 24C] (table 6.18), are designed for data sets s1c10 and s2c10.

Similar to the ‘Control-Set’ we designed for homogeneous container problem, we consider it worth designing heterogeneous containers problem’s own ‘Control-Set’ to check if both of our revised algorithms are also suitable for solving the new problem. Under our ‘Control-

	Reality Cost				Compromised Cost				Staged Cost			
s1c5b5	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	5	5	60	60	6	5.6	96	95.6	8	10.6	137	133.7
2	6	6	72	72	6	6	114	114	14	18	160.5	156.7
3	6	5.8	61	60.8	6	6	92	92	7	7.8	124	123.6
4	7	5.4	67	60.4	7	6	98	92	8	9	130.5	125.7
5	6	5.4	61	60.4	6	6	92	92	6	6.6	125	124.7
6	6	5.4	62	60.8	7	6.8	98	97	8	8	130	130
7	6	6	67	66.6	7	6.6	107	104	9	13.8	143	139.8
8	7	6.4	78	67	7	7.6	120	104.6	9	12	163.5	146.5
9	6	6	67	67	7	6.4	107	105.4	15	14.2	145	140.5
10	7	6	73	67	7	6	111	105	15	14	151.5	145.5
Ave	6.2	5.74	66.8	64.2	6.6	6.3	103.5	100.16	9.9	11.4	141	136.67
s1c5b10	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	6	6	67	67	7	7	107	106.6	12	13	140.5	139.9
2	5	5	60	60	7	6.4	98	93.4	9	11.2	129.5	125.4
3	6	6	72	72	7	6.6	115	114.6	12	13.6	161	160.3
4	8	6.6	79	72.8	8	8	112	112	15	17	152	150.4
5	6	6	67	66.2	7	7	106	106	12	14.8	148	141.5
6	6	5	66	60	7	7.8	102	98.2	9	10.6	136.5	131.3
7	5	5	60	60	5	5	95	95	11	11.4	134.5	132.8
8	6	6	66	66	6	6	101	101	11	11	141.5	136.3
9	6	5.6	61	60.6	6	6	92	92	10	13	129	126.8
10	6	5.8	61	60.8	6	6	92	92	7	9.2	124	122.8
Ave	6	5.7	65.9	64.54	6.6	6.58	102	101.08	10.8	12.48	139.65	136.75
s1c5b25	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	6	6	67	66.8	8	7.6	108	105.4	12	14.2	140.5	139.1
2	6	6	66	66	6	6	101	101	12	11.8	140.5	135.4
3	5	5	60	60	6	5.8	96	95.8	8	10	137	130.3
4	6	5	66	60	6	5	101	95	9	13.8	136.5	132.7
5	7	6	72	66	7	6	107	101	9	10.8	143	139.1
6	6	5	66	60	6	5	101	95	11	13.2	142	136.1
7	6	6	67	66	7	6	107	101	13	15	140	137
8	5	5	60	60	5	5	95	95	6	8.6	132	130.1
9	6	6	61	61	6	6	92	92	8	9.2	123.5	122.8
10	5	5	60	60	6	6	96	96	7	9.2	131	129.6
Ave	5.8	5.5	64.5	62.58	6.3	5.84	100.4	97.72	9.5	11.58	136.6	133.22
s1c10b5	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	11	11	127	127	11	11	200	200	15	18.8	275.5	272.8
2	11	11	122	122	12	12	193	193	19	26.2	264.5	260.3
3	12	12.6	139	136.8	13	12.6	221	214.8	18	23.2	308	290.1
4	13	12.8	146	145.4	14	14	231	228.8	18	18.8	314.5	314
5	12	12	138	138	13	13	216	216	32	31.4	295.5	293.3
6	11	11	126	126	11	11	196	196	12	12	270.5	270.5
7	12	10.6	127	120.6	12	11	193	187	11	11.4	257.5	257.2
8	13	12.8	151	145.6	14	15.6	240	234	17	21.4	334.5	321
9	13	13	144	140.6	13	13.8	221	217.8	22	24	299	297
10	12	12.2	139	138.4	13	14.4	221	215.4	27	27.2	301	295.4
Ave	12	11.9	135.9	134.04	12.6	12.84	213.2	210.28	19.1	21.44	292.05	287.16
s1c10b10	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	12	11.6	133	132.6	12	12	206	206	23	26.2	283	280.1
2	11	11	127	127	12	11.8	202	201.2	19	23	278	274
3	12	11.8	134	133.6	13	12.6	212	210.2	17	19.2	294	292.6
4	12	12	138	138	12	12	215	215	15	19.8	295.5	294.5
5	11	11	126	126	11	11	196	194	22	23.8	276	272.3
6	11	10	121	115	11	10	187	181	24	23.4	254	247.9
7	11	10	126	120	12	11.8	197	191.8	20	22.8	271	259.6
8	13	13	145	145	13	13	225	225	30	31.8	310.5	308
9	12	12.2	134	133.2	13	12.2	212	205.2	17	22.6	294	281.5
10	11	11	126	126	11	11	196	196	18	18.4	273	272.7
Ave	11.6	11.36	131	129.64	12	11.74	204.8	202.54	20.5	23.1	282.9	278.32
s1c10b25	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	12	11	133	126.8	12	11	206	200	21	24.4	283.5	274.5
2	11	10.8	121	119	11	10.8	187	184.8	21	22.2	256	252.8
3	11	11	132	132	13	13	207	207	22	24.2	283.5	280.5
4	11	11	127	126.8	11	11.8	200	196.8	18	19	280	272.4
5	12	11.6	134	131.6	14	14	213	210.8	15	16.6	288.5	282.1
6	13	12.4	139	132.2	13	12.8	212	204.8	18	20.8	286.5	275
7	11	11	127	127	11	11	200	200	19	22.8	285.5	273.2
8	12	11	138	132	13	13	216	216	22	21.6	297	290.7
9	11	11	121	121	11	10.6	187	184.6	18	19.2	253.5	252.6
10	11	11.2	126	124	11	11.2	196	195.2	21	21.2	270.5	269
Ave	11.5	11.2	129.8	127.24	12	11.92	202.4	200	19.5	21.2	278.45	272.28

Table 6.8 Iterated Local Search results for Unlimited Heterogeneous Containers Problem on data set soton1

	Reality Cost				Compromised Cost				Staged Cost			
	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
s2c5b5	6	6	61	61	6	6	92	92	20	14.4	120	120
1	6	5.8	61	60.8	6	6	92	92	18	18	122	122
2	7	5	67	60	7	6	98	92	14	16.4	125	123.6
3	6	5.4	61	60.4	6	6	92	92	17	17.8	122.5	122.1
4	6	5.6	61	60.6	6	6	92	92	21	18.2	126	121.8
5	6	5.8	61	60.8	6	6	92	92	8	12.4	123.5	120.3
6	5	5	55	55	5	5	86	86	7	6	124	118
7	6	5.8	61	60.8	6	6	92	92	14	12.8	125	120.8
8	6	5.2	62	60.4	7	6.4	98	96.8	17	13	129.5	126.6
9	6	5.8	61	60.8	6	6	92	92	20	20	120	120
10	6	5.54	61.1	60.06	6.1	5.94	92.6	91.88	15.6	14.9	123.75	121.52
Ave												
s2c5b10	6	5.8	61	60.8	6	6	92	92	15	13.6	124.5	121.9
1	5	5	55	55	5	5	86	86	12	12.2	120	119.9
2	6	6	67	67	7	6.4	107	105.6	12	12.8	147.5	147.1
3	6	5.6	61	60.6	6	6	92	92	16	16.8	123.5	121.6
4	6	5.8	61	60.8	6	6	92	92	18	17	128.5	122.6
5	6	6	61	61	6	5.8	92	90.8	10	12.4	121.5	120.1
6	6	5.8	61	60.8	6	6	92	92	8	10.4	123.5	121.8
7	6	5.6	61	60.6	6	6	92	92	15	13.2	124.5	119.4
8	5	5	55	55	5	5	86	86	13	14	119	118.3
9	6	5.8	61	60.8	6	6	92	92	11	12.4	121	120.2
10	5.8	5.64	60.4	60.24	5.9	5.82	92.3	92.04	13	13.48	125.35	123.29
Ave												
s2c5b25	6	5.8	61	60.8	6	6	92	92	15	15	124.5	124.5
1	6	5.6	61	60.6	6	6	92	92	12	13.4	120	119.1
2	6	5.8	61	59.8	6	5.6	92	89.6	12	12.4	120.5	120.3
3	5	5	55	55	5	5	86	86	9	10.2	122.5	119.2
4	6	5.8	61	60.8	6	6	92	92	14	13.2	125.5	119.3
5	6	6	61	61	6	6	92	92	12	12.4	120	119.7
6	6	6	61	61	6	6	92	92	11	12	121	120.4
7	6	5.8	61	60.8	6	6	92	92	16	16	123.5	123.5
8	6	5.8	61	60.8	6	6	92	92	10	12.8	122	119.9
9	6	6	61	61	6	6	92	92	8	11	123.5	121.2
10	5.9	5.76	60.4	60.16	5.9	5.86	91.4	91.16	11.9	12.84	122.3	120.71
Ave												
s2c10b5	9	9	108	108	10	10	172	172	29	30	235.5	234.9
1	10	10	114	114	10	10	177	177	12	15	243	241.1
2	10	10	114	114	10	10	177	177	14	18.6	241.5	238.2
3	9	9	108	108	9	9	171	171	18	18.2	238	237.9
4	10	10	115	114.8	11	11	183	183	34	33.2	245	244.7
5	10	10	114	114	10	10	177	176	18	18	245	238.6
6	10	10	115	115	11	10.4	183	181.8	40	36	247	244
7	9	9	108	108	10	10	172	172	18	21.4	238	236
8	10	10	115	114.8	11	11	183	182.6	25	28.6	247.5	244.3
9	10	9.8	114	110.4	10	10.2	177	176.2	28	25.8	243.5	237.6
10	9.7	9.68	112.5	112.1	10.2	10.16	177.2	176.86	23.6	24.48	242.4	239.73
Ave												
s2c10b10	10	10	114	114	10	10	177	177	31	31.2	241	240.9
1	10	10	114	114	10	10	177	177	33	33.2	239	238.9
2	9	9	108	108	10	10	172	172	16	18.4	240	237.1
3	9	9	108	108	9	9	171	171	20	20.4	236.5	236.3
4	9	9	108	108	10	10	172	172	13	13	235.5	235.5
5	9	9	108	108	9	9	171	171	24	26.4	240	238.4
6	10	10	114	114	10	10	177	177	26	25.2	245	238.8
7	10	10	115	114.2	11	10.8	183	182.6	32	32	247	247
8	9	9	108	108	9	9	171	171	15	17.2	240.5	239.2
9	10	9.4	110	108.8	11	10.8	174	173.4	11	15	237	234.4
10	9.5	9.44	110.7	110.5	9.9	9.86	174.5	174.4	22.1	23.2	240.15	238.65
Ave												
s2c10b25	10	10	114	114	10	10	177	177	18	18	245	238.5
1	10	10	114	114	10	10	177	177	21	21	242.5	242.5
2	10	10	114	114	10	10	177	177	18	18	245	242.4
3	10	10	114	114	10	10	177	177	20	19.8	243.5	239.8
4	10	10	114	114	10	10	177	177	26	26.4	245	244.7
5	10	10	114	114	10	10	177	177	14	16.4	241.5	239.9
6	12	11.8	133	132.8	13	13	207	207	30	30.6	282.5	281
7	10	10	114	114	10	10	177	177	21	21	249	243.6
8	10	10	114	114	10	10	177	177	22	22.6	241.5	241.1
9	10	10	114	114	10	10	177	177	29	29	242.5	242.5
10	10.2	10.18	115.9	115.88	10.3	10.3	180	180	21.9	22.28	247.8	245.6
Ave												

Table 6.9 Iterated Local Search results for Unlimited Heterogeneous Containers Problem on data set soton2

	2A 5B 12C				3A 6B 12C			
s1c5b5	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	10	10	77	77	8	7.2	70	65.2
2	14	14	101	101	9	9	78	78
3	8	8	65	65	7	7	63	63
4	9	8.4	71	67.4	8	7	69	63.2
5	9	9	71	71	8	7	69	63
6	10	10	77	77	7	7	64	64
7	12	11	89	83	8	8	71	71
8	17	13	119	95	12	8	96	70.2
9	12	11	89	83	8	8	71	71
10	13	12	95	89	9	8.6	77	74.6
Ave	11.4	10.64	85.4	80.84	8.4	7.68	72.8	68.32
s1c5b10	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	12	11	89	83	8	8	71	71
2	9	8.4	71	67.4	7	7	64	63
3	15	14.2	107	102.2	10	9	84	78
4	13	13	95	95	10	9	83	77.4
5	12	11	89	83	8	8	71	70.4
6	11	9.4	83	73.4	8	7.4	70	65.6
7	10	9.8	77	75.8	8	7	70	64
8	10	10	77	77	8	8	70	70
9	8	8	65	65	7	7	63	63
10	8	8	65	65	7	7	63	63
Ave	10.8	10.28	81.8	78.68	8.1	7.74	70.9	68.54
s1c5b25	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	11	11	83	83	8	8	71	71
2	10	10	77	77	8	8	70	70
3	10	9.2	77	72.2	7	7	64	64
4	10	10	77	77	8	7	70	64
5	12	11	89	83	8	8	71	71
6	11	10	83	77	8	8	70	70
7	11	11	83	82.8	8	8	71	70
8	10	9.2	77	72.2	8	7	70	64
9	9	8	71	65	7	7	63	63
10	10	9	77	71	7	7	64	64
Ave	10.4	9.84	79.4	76.02	7.7	7.5	68.4	67.1
s2c5b5	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	8	8	65	65	7	7	63	63
2	8	8	65	65	7	7	63	63
3	9	9	71	71	7	7	64	63
4	9	8	71	65	7	7	63	63
5	8	8	65	65	7	7	63	63
6	8	8	65	65	7	7	63	63
7	8	8	65	65	6	6	57	57
8	8	8	65	65	7	7	63	63
9	9	9	71	71	7	7	64	64
10	8	8	65	65	7	7	63	63
Ave	8.3	8.2	66.8	66.2	6.9	6.9	62.6	62.5
s2c5b10	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	8	8	65	65	7	7	63	63
2	8	8	65	63.6	7	6.8	63	61.8
3	12	12	89	89	9	8	77	71
4	8	8	65	65	7	7	63	63
5	9	8	71	65	7	7	63	63
6	8	8	65	65	7	7	63	63
7	8	8	65	65	7	7	63	63
8	8	8	65	65	7	7	63	63
9	8	7.4	65	61.4	7	6.2	63	58.2
10	8	8	65	65	7	7	63	63
Ave	8.5	8.34	68	66.9	7.2	7	64.4	63.2
s2c5b25	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	9	8.8	71	69.8	8	7.2	69	63.4
2	8	8	65	65	7	7	63	60.8
3	8	8	58	58	7	7	63	63
4	7	7	59	59	6	6	57	57
5	8	8	65	65	7	7	63	63
6	8	8	65	65	7	7	63	63
7	8	8	65	65	7	7	63	60.6
8	9	8.4	71	67.4	7	7	63	63
9	8	8	65	65	7	7	63	63
10	8	8	65	65	7	7	63	63
Ave	8.1	8.02	64.9	64.42	7	6.92	63	61.98

Table 6.10 Iterated Local Search results for Limited Heterogeneous Containers Problem on data sets s1c5 and s2c5

	4A 10B 24C				6A 12B 24C			
s1c10b5	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	22	20.2	166	155.2	15	15	135	135
2	19	18.6	148	145.6	14	14	128	128
3	27	23	196	172	17	16	149	141.8
4	29	28	208	202	19	18	162	156
5	24	24	178	178	18	17	154	148.4
6	22	20.6	166	157.6	15	15	135	134
7	20	19	154	148	14	14	128	128
8	31	27.8	220	200.8	21	18	174	155.4
9	26	25.2	190	185.2	18	17	154	148
10	25	24	184	178	17	16.6	148	145.6
Ave	24.5	23.04	181	172.24	16.8	16.06	146.7	142.02
s1c10b10	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	23	22	172	166	16	16	142	141
2	21	21	160	160	16	15	141	135
3	25	25	184	184	17	17	148	148
4	26	25	190	184	17	17	149	148.2
5	20	20	154	154	15	15	134	134
6	18	17	142	136	15	13.4	133	123.4
7	20	18.4	154	144.4	16	14	140	127.8
8	28	26	202	190	18	18	156	155
9	22	21.4	166	162.4	16	16	142	141
10	22	20.4	166	156.4	15	15	134	134
Ave	22.5	21.62	169	163.72	16.1	15.64	141.9	138.74
s1c10b25	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	23	22	172	166	16	15	142	134.6
2	18	17.4	142	138.4	14	14	128	127
3	24	22.4	178	168.4	17	16	148	142
4	21	20	160	154	16	15	141	135
5	24	22.8	178	170.8	17	16	148	142
6	23	20.8	172	158.8	16	15.4	142	136.4
7	23	21	172	160	16	15	141	135
8	25	24	184	178	17	17	149	148
9	18	17	142	136	14	14	127	126.2
10	20	19.6	154	151.6	15	15	134	134
Ave	21.9	20.7	165.4	158.2	15.8	15.24	140	136.02
s2c10b5	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	15	14.2	124	119.2	12	12	114	114
2	15	15	124	124	13	13	120	120
3	16	15	130	124	13	13	120	120
4	15	14.6	124	121.6	13	12.4	120	115.6
5	16	16	130	130	13	13	121	121
6	15	15	124	124	13	13	120	120
7	17	16	136	130	13	13	121	121
8	15	14.2	124	119.2	12	12	114	114
9	16	16	130	130	13	13	121	121
10	16	14.8	130	122.8	13	12.8	120	118.8
Ave	15.6	15.08	127.6	124.48	12.8	12.72	119.1	118.54
s2c10b10	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	15	15	124	124	13	13	120	120
2	16	15	130	124	13	13	120	120
3	14	14	118	118	12	12	114	114
4	15	15	124	124	13	12.8	120	118
5	15	14.2	124	119.2	12	12	114	114
6	15	15	124	124	13	13	120	120
7	16	15	130	124	13	13	120	120
8	16	16	130	130	13	13	121	121
9	15	14.2	124	119.2	13	13	120	119.2
10	15	14	124	118	12	12	114	114
Ave	15.2	14.74	125.2	122.44	12.7	12.68	118.3	118.02
s2c10b25	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost	Initial Cont No.	Best Cont No.	Initial Cost	Best Cost
1	15	15	124	124	13	13	120	120
2	15	15	124	124	13	13	120	120
3	16	15	130	124	13	13	120	120
4	16	15	130	124	13	13	120	120
5	16	16	130	130	14	13.6	126	123.8
6	15	15	124	124	13	13	120	120
7	22	22	166	166	16	16	142	141.6
8	16	16	130	130	13	13	121	121
9	15	15	124	124	13	13	120	120
10	16	16	130	130	13	13	120	120
Ave	16.2	16	131.2	130	13.4	13.36	122.9	122.64

Table 6.11 Iterated Local Search results for Limited Heterogeneous Containers Problem on data sets s1c10 and s2c10

Set' implementation, all boxes are placed within their original batch, and all the batches (box types) are sorted into a random order. Cargo Manager (CM) is then repeatedly run until all boxes are pack, and the total container cost is calculated. Among all available container types, a container type is randomly selected every time when CM runs. The same process is repeated till a set time, and the lowest total container cost is selected as the best result. Each instance is run 5 times, and the average is taken as the final result. It is important to point out that we only implement 'Control-Set' under staged cost for unlimited heterogeneous containers problem. Because results from BS and ILSSN show that most packed containers are 40ft container type (A) due to its cost advantage under reality and compromised cost, the results of 'Control-Set' are not likely to hold any value for comparison when all container types have equal chance to be selected. For each instance, 'Control-Set' is given the same running time shown in table 6.14 as BS and ILSSN. Its results are compared with those of BS and ILSSN. Only better results (i.e. lower cost) from algorithm(s) serve as the proof of a working BS and/or ILSSN.

Table 6.12 presents the results comparison of 'Control-Set', beam search (BS) and iterated local search with simple neighbourhood (ILSSN) under staged cost. The best result among the three is bolded for each instance and average result of each data set. In general, Beam search yields the best results with the exception of 'Control-Set' having the best average for data set s1c10b5 and ILSSN having the best average results for data sets s2c5b5 and s2c10b5. 'Control-Set' comes second beating ILSSN on data sets s1c5b5, s2c5b10 and s2c10b25. Based on the comparison, we would consider that both of our algorithms have merits when dealing with heterogeneous containers problem. The reason that 'Control-Set' seemingly performs better in some occasions is because having cost set as objective diminishes the relevance of utilisation while both algorithms are heavily built on the concept of utilisation improvement.

Table 6.13 presents the results comparison of beam search and ILSSN under compromised cost. Comparison shows that beam search generates better results in general. Of all 12 data sets, ILSSN only generates better average results on 4 data sets, s1c5b5, s1c5b25, s1c10b5 and s1c10b10. Table 6.14 presents the results comparison of beam search and ILSSN under reality cost. Comparison shows that ILSSN generates better results in general. Of all 12 data sets, beam search only generates better average results on 4 data sets, s2c5b25, s2c10b5, s2c10b10 and s2c10b25. Table 6.15 presents the results comparison of beam search and ILSSN under setting [2A 5B 12C]. Among 6 data sets, beam search generates better average results on 2 data sets, s1c5b10 and s2c5b25, while beam search and ILSSN tie on data set s2c5b5. Table 6.16 presents the results comparison of beam search and ILSSN under

setting [3A 6B 12C]. Among 6 data sets, beam search generates better average results on 2 data sets, s1c5b25 and s2c5b25. Table 6.17 presents the results comparison of beam search and ILSSN under setting [4A 10B 24C]. Among 6 data sets, beam search generates better average results on 2 data sets, s1c10b25 and s2c10b25. Table 6.18 presents the results comparison of beam search and ILSSN under setting [6A 12B 24C]. Among 6 data sets, beam search generates better average results on all three soton2 data sets. Apart from results under compromised cost in which beam search still has an upper hand, all the rest results comparisons under reality cost show mixed messages with ILSSN generating more better average results. However, it is still hard to say which algorithm is better. If we investigate closer we will see that even if for a certain data set an algorithm has better average result, it might be caused by it doing extremely well on only a few instances with slightly worse results on all the rest instances.

s1c5b5				s1c5b10				s1c5b25			
Instance	Ctrl-Set	BS	ILS	Instance	Ctrl-Set	BS	ILS	Instance	Ctrl-Set	BS	ILS
1	132	131	133.7	1	141.8	138.5	139.9	1	142.1	138.2	139.1
2	156	156.5	156.7	2	127.4	123.7	125.4	2	137.3	134.5	135.4
3	124.1	123.1	123.6	3	159.5	156.6	160.3	3	134.2	127.3	130.3
4	125.7	124.4	125.7	4	151.1	149.9	150.4	4	133.1	131.5	132.7
5	124.4	124	124.7	5	139.9	140.3	141.5	5	144.2	137.9	139.1
6	130.3	130.3	130	6	129.8	127.6	131.3	6	136.8	133.7	136.1
7	141.2	139.3	139.8	7	134.6	130.2	132.8	7	138.3	134.6	137
8	137.6	127.4	146.5	8	135.9	135.2	136.3	8	133.2	130	130.1
9	141.5	141.4	140.5	9	124.6	123	126.8	9	126.7	122.5	122.8
10	147.4	146.2	145.5	10	124.1	122	122.8	10	134.6	128.7	129.6
Ave.	136.02	134.36	136.67	Ave.	136.87	134.7	136.75	Ave.	136.05	131.89	133.22

s1c10b5				s1c10b10				s1c10b25			
Instance	Ctrl-Set	BS	ILS	Instance	Ctrl-Set	BS	ILS	Instance	Ctrl-Set	BS	ILS
1	274.1	272.5	272.8	1	283.5	280.1	280.1	1	273.7	268.5	274.5
2	260.7	263.4	260.3	2	276.3	274.2	274	2	257.6	254.2	252.8
3	285.6	292.4	290.1	3	290.9	300.6	292.6	3	290.9	281.3	280.5
4	313.6	325.7	314	4	297.6	291.9	294.5	4	278.5	268.7	272.4
5	295.3	291.6	293.3	5	267.2	264.6	272.3	5	289.3	281.9	282.1
6	270.2	271.9	270.5	6	253.2	251.6	247.9	6	274.8	270.6	275
7	261.4	266.1	257.2	7	261.5	255.2	259.6	7	276.8	272.4	273.2
8	300.7	303.9	321	8	311.5	305.2	308	8	295.9	292.7	290.7
9	297	296.3	297	9	284.3	283.3	281.5	9	258.5	251.1	252.6
10	293.2	298.9	295.4	10	271.2	267.2	272.7	10	271.8	265.3	269
Ave.	285.18	288.27	287.16	Ave.	279.72	277.39	278.32	Ave.	276.78	270.67	272.28

s2c5b5				s2c5b10				s2c5b25			
Instance	Ctrl-Set	BS	ILS	Instance	Ctrl-Set	BS	ILS	Instance	Ctrl-Set	BS	ILS
1	120.2	119.2	120	1	122	122	121.9	1	122.8	124.2	124.5
2	122	122.1	122	2	119	118.1	119.9	2	121.9	119.4	119.1
3	124.6	124.3	123.6	3	143.1	145.5	147.1	3	122	119.6	120.3
4	123.5	122	122.1	4	120.8	120	121.6	4	121.1	118.5	119.2
5	123	122.4	121.8	5	122.4	121.3	122.6	5	120.7	119.7	119.3
6	120.9	120.7	120.3	6	121.1	120.1	120.1	6	120.4	118.5	119.7
7	118.5	119	118	7	122	121.2	121.8	7	122.4	120.4	120.4
8	120.9	120.6	120.8	8	120.1	119.4	119.4	8	122.3	122.5	123.5
9	127.1	127.3	126.6	9	117.9	117.5	118.3	9	120.5	119.6	119.9
10	120.8	120	120	10	121	120.1	120.2	10	122.5	121.6	121.2
Ave.	122.15	121.76	121.52	Ave.	122.94	122.52	123.29	Ave.	121.66	120.4	120.71

s2c10b5				s2c10b10				s2c10b25			
Instance	Ctrl-Set	BS	ILS	Instance	Ctrl-Set	BS	ILS	Instance	Ctrl-Set	BS	ILS
1	236	234.9	234.9	1	240.3	237.9	240.9	1	240.7	239.3	238.5
2	241.4	240.2	241.1	2	240.7	238.7	238.9	2	241.3	241.3	242.5
3	239.1	238.6	238.2	3	236.9	234.2	237.1	3	241	240.2	242.4
4	239.9	236.5	237.9	4	238.3	236.5	236.3	4	240.3	238.4	239.8
5	246.5	247.4	244.7	5	236.6	235.7	235.5	5	241.5	241.6	244.7
6	240.7	238.8	238.6	6	238.8	237.2	238.4	6	239.7	237.6	239.9
7	247.6	248.2	244	7	239.7	238.9	238.8	7	283.2	281.5	281
8	235.7	235.5	236	8	245.1	243.6	247	8	241.2	240	243.6
9	245.4	244.5	244.3	9	237.6	235.1	239.2	9	239.8	238.9	241.1
10	238.5	237.2	237.6	10	235.6	233.5	234.4	10	240.7	239	242.5
Ave.	241.08	240.18	239.73	Ave.	238.96	237.13	238.65	Ave.	244.94	243.78	245.6

Table 6.12 Results Comparison for Unlimited Heterogeneous Containers Problem under Staged Cost

s1c5b5			s1c5b10			s1c5b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	95	95.6	1	105	106.6	1	104.6	105.4
2	114.6	114	2	92	93.4	2	101	101
3	92	92	3	114	114.6	3	95	95.8
4	92	92	4	111	112	4	95	95
5	92	92	5	104.2	106	5	101	101
6	95.2	97	6	94.2	98.2	6	98.6	95
7	101.8	104	7	95	95	7	101	101
8	120.8	104.6	8	101	101	8	95	95
9	105	105.4	9	92	92	9	92	92
10	106.2	105	10	92	92	10	95.2	96
Ave.	101.46	100.16	Ave.	100.04	101.08	Ave.	97.84	97.72

s1c10b5			s1c10b10			s1c10b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	200	200	1	206	206	1	196.4	200
2	190.8	193	2	205	201.2	2	184.6	184.8
3	214	214.8	3	225	210.2	3	206.6	207
4	240.8	228.8	4	215.4	215	4	198.6	196.8
5	216	216	5	190.4	194	5	208.2	210.8
6	199	196	6	187	181	6	200.8	204.8
7	193.8	187	7	188	191.8	7	200	200
8	221.4	234	8	225	225	8	215.2	216
9	215.2	217.8	9	209.8	205.2	9	183.4	184.6
10	215.2	215.4	10	193.6	196	10	196.8	195.2
Ave.	210.62	210.28	Ave.	204.52	202.54	Ave.	199.06	200

s2c5b5			s2c5b10			s2c5b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	92	92	1	92	92	1	92	92
2	92	92	2	86	86	2	90.8	92
3	92	92	3	105	105.6	3	92	89.6
4	92	92	4	92	92	4	86	86
5	92	92	5	92	92	5	92	92
6	92	92	6	89.6	90.8	6	86	92
7	86	86	7	92	92	7	92	92
8	92	92	8	92	92	8	92.2	92
9	95.8	96.8	9	86	86	9	92	92
10	92	92	10	92	92	10	92	92
Ave.	91.78	91.88	Ave.	91.86	92.04	Ave.	90.7	91.16

s2c10b5			s2c10b10			s2c10b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	172	172	1	171	177	1	177	177
2	177	177	2	177	177	2	177	177
3	171	177	3	172	172	3	177	177
4	171	171	4	171.2	171	4	175.8	177
5	181.4	183	5	172	172	5	177	177
6	173.4	176	6	171	171	6	171	177
7	181	181.8	7	177	177	7	206	207
8	171.6	172	8	177	182.6	8	175.8	177
9	180.2	182.6	9	172.2	171	9	172.2	177
10	171	176.2	10	172.6	173.4	10	175.8	177
Ave.	174.96	176.86	Ave.	173.3	174.4	Ave.	178.46	180

Table 6.13 Results Comparison for Unlimited Heterogeneous Containers Problem under Compromised Cost

s1c5b5			s1c5b10			s1c5b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	60	60	1	67	67	1	67	66.8
2	72	72	2	60	60	2	66	66
3	60.4	60.8	3	72	72	3	60	60
4	61	60.4	4	72	72.8	4	60.4	60
5	60.6	60.4	5	67	66.2	5	66	66
6	62	60.8	6	60.6	60	6	64.8	60
7	66	66.6	7	60	60	7	66.4	66
8	78	67	8	66	66	8	60	60
9	67	67	9	60.6	60.6	9	61	61
10	68	67	10	61	60.8	10	60	60
Ave.	65.5	64.2	Ave.	64.62	64.54	Ave.	63.16	62.58

s1c10b5			s1c10b10			s1c10b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	127	127	1	132.4	132.6	1	126.8	126.8
2	120.8	122	2	129.4	127	2	120.2	119
3	136	136.8	3	143.2	133.6	3	132	132
4	156	145.4	4	138	138	4	126	126.8
5	138	138	5	120	126	5	132	131.6
6	130.2	126	6	121	115	6	128.8	132.2
7	120.4	120.6	7	120	120	7	127	127
8	140.2	145.6	8	144.8	145	8	138	132
9	138.2	140.6	9	133.6	133.2	9	117.4	121
10	139	138.4	10	121.2	126	10	126.2	124
Ave.	134.58	134.04	Ave.	130.36	129.64	Ave.	127.44	127.24

s2c5b5			s2c5b10			s2c5b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	60.4	61	1	61	60.8	1	60.8	60.8
2	60.6	60.8	2	55	55	2	59	60.6
3	61	60	3	70	67	3	60.2	59.8
4	61	60.4	4	60.4	60.6	4	55	55
5	61	60.6	5	61	60.8	5	60	60.8
6	60	60.8	6	60.8	61	6	55	61
7	55	55	7	61	60.8	7	61	61
8	61	60.8	8	60	60.6	8	61	60.8
9	60	60.4	9	55	55	9	60	60.8
10	61	60.8	10	60.4	60.8	10	61	61
Ave.	60.1	60.06	Ave.	60.46	60.24	Ave.	59.3	60.16

s2c10b5			s2c10b10			s2c10b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	108	108	1	109.2	114	1	114	114
2	114	114	2	114	114	2	114	114
3	108	114	3	108	108	3	114	114
4	108	108	4	108	108	4	114	114
5	115	114.8	5	108	108	5	114	114
6	110.4	114	6	108	108	6	108.4	114
7	115	115	7	114	114	7	133	132.8
8	108	108	8	114	114.2	8	113.2	114
9	115.8	114.8	9	108	108	9	109.2	114
10	108	110.4	10	110	108.8	10	112.8	114
Ave.	111.02	112.1	Ave.	110.12	110.5	Ave.	114.66	115.88

Table 6.14 Results Comparison for Unlimited Heterogeneous Containers Problem under Reality Cost

s1c5b5			s1c5b10			s1c5b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	77	77	1	83	83	1	83	83
2	101	101	2	65	67.4	2	80.6	77
3	65	65	3	104.6	102.2	3	71	72.2
4	71	67.4	4	95	95	4	77	77
5	71	71	5	83	83	5	83	83
6	77	77	6	71	73.4	6	77	77
7	83	83	7	77	75.8	7	83	82.8
8	114	95	8	77	77	8	77	72.2
9	83	83	9	65	65	9	65	65
10	95	89	10	65	65	10	74.6	71
Ave.	83.7	80.84	Ave.	78.56	78.68	Ave.	77.12	76.02

s2c5b5			s2c5b10			s2c5b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	65	65	1	65	65	1	65	69.8
2	65	65	2	65	63.6	2	65	65
3	71	71	3	89	89	3	65	58
4	65	65	4	65	65	4	59	59
5	65	65	5	65	65	5	65	65
6	65	65	6	65	65	6	63.8	65
7	65	65	7	65	65	7	65	65
8	65	65	8	65	65	8	65	67.4
9	71	71	9	61.4	61.4	9	65	65
10	65	65	10	65	65	10	65	65
Ave.	66.2	66.2	Ave.	67.04	66.9	Ave.	64.28	64.42

Table 6.15 Results Comparison for Limited Heterogeneous Containers Problem under Container Setting of 2A 5B 12C

s1c5b5			s1c5b10			s1c5b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	70	65.2	1	71	71	1	71	71
2	78	78	2	63	63	2	70	70
3	63	63	3	80.4	78	3	64	64
4	63	63.2	4	77.4	77.4	4	64	64
5	63	63	5	71	70.4	5	70.8	71
6	64	64	6	64	65.6	6	70	70
7	71	71	7	64	64	7	70	70
8	87.4	70.2	8	70	70	8	64	64
9	71	71	9	63	63	9	63	63
10	77	74.6	10	63	63	10	64	64
Ave.	70.74	68.32	Ave.	68.68	68.54	Ave.	67.08	67.1

s2c5b5			s2c5b10			s2c5b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	63	63	1	63	63	1	63	63.4
2	63	63	2	63	61.8	2	63	60.8
3	63.4	63	3	74.4	71	3	63	63
4	63	63	4	63	63	4	57	57
5	63	63	5	63	63	5	63	63
6	63	63	6	63	63	6	57	63
7	57	57	7	63	63	7	63	60.6
8	63	63	8	63	63	8	63	63
9	64	64	9	57	58.2	9	63	63
10	63	63	10	63	63	10	63	63
Ave.	62.54	62.5	Ave.	63.54	63.2	Ave.	61.8	61.98

Table 6.16 Results Comparison for Limited Heterogeneous Containers Problem under Container Setting of 3A 6B 12C

s1c10b5			s1c10b10			s1c10b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	158.8	155.2	1	167.2	166	1	154	166
2	148	145.6	2	161.2	160	2	142	138.4
3	181.6	172	3	196	184	3	166	168.4
4	203.2	202	4	185.2	184	4	154	154
5	178	178	5	151.6	154	5	168.4	170.8
6	158.8	157.6	6	138.4	136	6	156.4	158.8
7	162.2	148	7	144.4	144.4	7	163.6	160
8	192.4	200.8	8	194.8	190	8	179.2	178
9	184	185.2	9	167.2	162.4	9	138.4	136
10	188.8	178	10	154	156.4	10	155.2	151.6
Ave.	175.58	172.24	Ave.	166	163.72	Ave.	157.72	158.2

s2c10b5			s2c10b10			s2c10b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	118	119.2	1	124	124	1	124	124
2	124	124	2	124	124	2	127.6	124
3	124	124	3	118	118	3	124	124
4	124	121.6	4	124	124	4	124	124
5	132.4	130	5	118	119.2	5	127.6	130
6	124	124	6	124	124	6	124	124
7	131.2	130	7	124	124	7	166	166
8	118	119.2	8	130	130	8	126.4	130
9	130	130	9	124	119.2	9	124	124
10	124	122.8	10	118	118	10	124	130
Ave.	124.96	124.48	Ave.	122.8	122.44	Ave.	129.16	130

Table 6.17 Results Comparison for Limited Heterogeneous Containers Problem under Container Setting of 4A 10B 24C

s1c10b5			s1c10b10			s1c10b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	135	135	1	141.8	141	1	134.2	134.6
2	130.4	128	2	141	135	2	127	127
3	146.8	141.8	3	154.2	148	3	141.4	142
4	160.8	156	4	149	148.2	4	134.2	135
5	148.4	148.4	5	132.8	134	5	142.8	142
6	140.8	134	6	127	123.4	6	137	136.4
7	135	128	7	127.8	127.8	7	135	135
8	153.8	155.4	8	155.8	155	8	148.2	148
9	148.8	148	9	143	141	9	125.8	126.2
10	150	145.6	10	134	134	10	136.2	134
Ave.	144.98	142.02	Ave.	140.64	138.74	Ave.	136.18	136.02

s2c10b5			s2c10b10			s2c10b25		
Instance	BS	ILS	Instance	BS	ILS	Instance	BS	ILS
1	114	114	1	120	120	1	120	120
2	120	120	2	120	120	2	120	120
3	120	120	3	114	114	3	120	120
4	118.8	115.6	4	116.4	118	4	120	120
5	121	121	5	114	114	5	120	123.8
6	120	120	6	120	120	6	120	120
7	121	121	7	120	120	7	143	141.6
8	114	114	8	120.6	121	8	120	121
9	121	121	9	114	119.2	9	120	120
10	115.2	118.8	10	114	114	10	120	120
Ave.	118.5	118.54	Ave.	117.3	118.02	Ave.	122.3	122.64

Table 6.18 Results Comparison for Limited Heterogeneous Containers Problem under Container Setting of 6A 12B 24C

A short conclusion is drawn for the comparison session. Both beam search and iterated local search with simple neighbourhood are able to outperform ‘Control-Set’ on most instances. Thus, we consider that both revised algorithms are suitable for solving heterogeneous containers problem. In general, beam search performs better compared to ILSSN under staged and compromised cost. This is expected as solving heterogeneous containers problem under staged cost somewhat assembles solving homogeneous container problem, and beam search

algorithm outperforms ILSSN on homogeneous container problem. On the other hand, it is hard to tell which algorithm performs better under reality cost. The fact that reality cost makes improvement on utilisation less relevant to the final result causes both algorithms to perform less effectively and less consistently under such cost setting.

6.6 Conclusion

In this chapter, we modify beam search and iterated local search with simple neighbourhood algorithms, which are originally designed for homogeneous container problem, to adapt to heterogeneous containers problem. We then simulate a real industry situation by setting up a combination of available container types along with three cost settings ranging from the reality cost to the more scientific one. Results suggest that beam search performs better than ILSSN under staged price which is proportional to container volume, as utilisation is still a major factor in deciding the best solution. Based on the results, it is hard to tell whether BS or ILSSN performs better under reality cost. Each method generates about half of the best results. The cause of it is that under reality cost larger container types are preferred during packing because of the cost advantage. Therefore, a less well-packed larger container may be preferred over a well-packed smaller container. This also makes utilisation less relevant in decision making. Similar to homogeneous container problem in chapter 5, results comparison among the two algorithms and Control-Set is made. Comparison suggests that beam search works well for almost all the data sets while ILSSN has its limit on a few data sets especially on those have fewer box types.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis solves multiple containers problem which directly tackles an end industrial problem where the large scale of container shipping happens on a daily basis. This is achieved by embedding a single container constructive heuristic in our improvement heuristics. While the majority of the research focus on single container problem, the research focus however might contradict the need of industry. Utilisation is to be maximised for single container problem. While utilisation still remains as a large impact on the final solution of multiple containers problem, a well packed container with high utilisation may be responsible for the mediocre packing in the rest containers and may be the cause of using extra container(s). By embedding a single container constructive heuristic which is excellent in packing single container, we firstly guarantee the solution quality of each single container. Also, the improvement heuristics are able to see the big picture by excluding packings which might drag down performances of other containers. Another striking feature is that the solutions generated by our improvement heuristics is not only theoretically feasible but also of industry standard.

Our experimental results indicate that improvement algorithm with simple structure and less intervention performs better when constructive heuristic is less controllable. We designed four improvement algorithms for homogeneous container problem: iterated local search with intelligent neighbourhood (ILSIN), iterated local search with descent and randomness (ILSDR), iterated local search with simple neighbourhood (ILSSN) and beam search (BS). In general, results from BS are better than those from ILSSN while ILSSN performs better

than ILSIN. Beam search allows the constructive heuristic to do what it is good at, packing single containers, without any intervention. Beam search only makes decision on the selection of partial solutions, i.e. packed single containers. The framework of BS guides through each selection step and eventually leads to the final solution. Compared to BS, ILSSN intervenes constructive algorithm by trying to correct constructive heuristic's 'mistake'. Despite having a rather simple structure and a goal-oriented design, repacking what constructive heuristic has already packed does not give ILSSN an edge over BS. Although ILSIN has a similar structure towards ILSSN, it exploits problem specific knowledge in order to generate fewer but more promising packing patterns within the same neighbourhood. More intervention under ILSIN is performed, including partitioning single container into a few packing spaces and constantly evaluating the fitness among box types with newly updated box number. However, such complexity and effort do not make ILSIN outperform its cousin algorithm ILSIN, let alone BS. Therefore, we are convinced that a simple, goal-oriented improvement algorithm with little intervention is best suitable for the situation where the relationship between permutation and packing remains not straightforward. On the other note, beam search is implemented on the three-dimensional container loading problem for the first time.

While heterogeneous containers problem has never been the focus of research as we discussed in our literature review in chapter 2, the outcome of our experiments may provide us some insights on this issue. In reality, containers are assigned with costs in a way that larger container has a clear cost advantage. As we pointed in chapter 6, 40ft container has a cost of 12 while 20ft and 10ft container has a cost of 7 and 6 respectively. Such cost advantage on 40ft container makes the utilisation and actually packing the least impact. Decision can be made without any mathematical analysis. For unlimited heterogeneous containers problem, 40ft container will always be the first choice except for the last pack container where 20ft or 10ft container may be used if all remaining boxes can be packed in it. The largest container will still be the first choice for limited heterogeneous containers problem, with 20ft container the second and 10ft container the last choice. Hence, it is not surprising that such real-industry costs greatly limit the research opportunities.

7.2 Future Work

While we chose iterated local search and beam search to tackle the specific problem we face, we must not forget that there are other methods out there suitable for solving our problem. It might be worth exploring those methods including the path of commercial solver. Also,

the research conducted in this thesis has not involved in any constraint requirements such as heavy and fragile boxes and multi-drop. Moreover, although not an field rarely touched any more it will be interesting to combine our multiple containers problem with vehicle routing problems, especially with constraints such as multi-drop.

The other potential work we can carry out is in heterogeneous containers problem. One aspect which distinguishes heterogeneous container problem from other 3D containers problem is the presence of different box types (with different box numbers for limited heterogeneous containers problem). However, our algorithms along with those from the literature seem to ignore this fact and simply extend algorithms originally designed for solving homogeneous container problem. It may be interesting to design a model reflecting this characteristic. For example, a heuristic model can be designed on swapping, inserting, adding and removing packed containers instead of doing it on boxes. However, we will have to set container cost proportional to container volume for any future investigation on this problem type to be worth.

References

- Allen, S.D., Burke, E.K., Kendall, G., 2011. A hybrid placement strategy for the three-dimensional strip packing problem. *European Journal of Operational Research* 209, 219-227.
- Allen, S.D., Burke, E.K., Mareček, J., 2012. A space-indexed formulation of packing boxes into a larger box. *Operations Research Letters* 40, 20-24.
- Alonso, M.T., Valdes, R.A., Tamarit, J.M., Parreño, F., 2014. A reactive GRASP algorithm for the container loading problem with load-bearing constraints. *European Journal of Industrial Engineering* 8, 669-694.
- Alvarez-Valdes, R., Parreño, F., Tamarit, J.M., 2013a. A GRASP/Path Relinking algorithm for two- and three-dimensional multiple bin-size bin packing problems. *Computers & Operations Research* 40, 3081-3090.
- Alvarez-Valdes, R., Parreño, F., Tamarit, J.M., 2013b. Lower bounds for three-dimensional multiple-bin-size bin packing problems. *OR Spectrum* 37, 49-74.
- Amossen, R.R., Pisinger, D., 2010. Multi-dimensional bin packing problems with guillotine constraints. *Computers & Operations Research* 37, 1999-2006.
- Araya, I., Riff, M.-C., 2014. A beam search approach to the container loading problem. *Computers & Operations Research* 43, 100-107.
- Baldi, M.M., Crainic, T.G., Perboli, G., Tadei, R., 2012. The generalized bin packing problem. *Transportation Research Part E: Logistics and Transportation Review* 48, 1205-1220.
- Baldi, M.M., Crainic, T.G., Perboli, G., Tadei, R., 2014. Branch-and-price and beam search algorithms for the Variable Cost and Size Bin Packing Problem with optional items. *Annals of Operations Research* 222, 125-141.

- Baum, E.B., 1986. Iterated descent: a better algorithm for local search in combinatorial optimization problems. Technical report, Caltech, Pasadena, CA. Manuscript.
- Baxter, J., 1981. Local Optima Avoidance in Depot Location. *The Journal of the Operational Research Society* 32, 815-819.
- Bengtsson, B.-E., 1982. Packing Rectangular Pieces-A Heuristic Approach. *The Computer Journal* 25, 353-357.
- Bennell, J.A., Song, X., 2008. A beam search implementation for the irregular shape packing problem. *Journal of Heuristics* 16, 167-188.
- Bischoff, E.E., 2006. Three-dimensional packing of items with limited load bearing strength. *European Journal of Operational Research* 168, 952-966.
- Bischoff, E.E., Marriott, M.D., 1990. A comparative evaluation of heuristics for container loading. *European Journal of Operational Research* 44, 267-276.
- Bischoff, E.E., Ratcliff, M.S.W., 1995. Issues in the development of approaches to container loading. *Omega* 23, 377-390.
- Blum, C., 2005. Beam-ACO-hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers & Operations Research* 32, 1565-1591.
- Bortfeldt, A., 2000. Eine Heuristik für Multiple Containerladeprobleme. *OR Spektrum* 22, 239-261.
- Bortfeldt, A., Gehring, H., 1998. Applying Tabu Search to Container Loading Problems. Fernuniversität Hagen - Lehrstuhl für Wirtschaftsinformatik, Prof. Gehring - Publikationen; In: *Operations Research Proceedings, 1997*, Springer Verlag, Berlin u.a. 1998, S. 533-538.
- Bortfeldt, A., Gehring, H., 2001. A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research* 131, 143-161.
- Bortfeldt, A., Gehring, H., Mack, D., 2003. A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing* 29, 641-662.
- Bortfeldt, A., Mack, D., 2007. A heuristic for the three-dimensional strip packing problem. *European Journal of Operational Research* 183, 1267-1279.
- Bortfeldt, A., Wäscher, G., 2013. Constraints in container loading – A state-of-the-art review. *European Journal of Operational Research* 229, 1-20.

- Boschetti, M.A., 2004. New lower bounds for the three-dimensional finite bin packing problem. *Discrete Applied Mathematics* 140, 241-258.
- Bożejko, W., Kacprzak, Ł., Wodecki, M., 2015. Parallel Coevolutionary Algorithm for Three-Dimensional Bin Packing Problem, in: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (Eds.), *Artificial Intelligence and Soft Computing, Lecture Notes in Computer Science*. Springer International Publishing, pp. 319-328.
- Brunetta, L., Grégoire, P., 2005. A General Purpose Algorithm for Three-Dimensional Packing. *INFORMS Journal on Computing* 17, 328-338.
- Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J., 2012. Automating the packing heuristic design process with genetic programming. *Evolutionary Computation* 20, 63-89.
- Černý, V., 1985. A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications* 45, 41-51.
- Ceschia, S., Schaerf, A., 2013. Local search for a multi-drop multi-container loading problem. *Journal of Heuristics* 19, 275-294.
- Che, C.H., Huang, W., Lim, A., Zhu, W., 2011a. The multiple container loading cost minimization problem. *European Journal of Operational Research* 214, 501-511.
- Che, C.H., Huang, W., Lim, A., Zhu, W., 2011b. A Heuristic for the Multiple Container Loading Cost Minimization Problem, in: Mehrotra, K.G., Mohan, C.K., Oh, J.C., Varshney, P.K., Ali, M. (Eds.), *Modern Approaches in Applied Intelligence, Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 276-285.
- Chen, C.S., Lee, S.M., Shen, Q.S., 1995. An analytical model for the container loading problem. *European Journal of Operational Research* 80, 68-76.
- Chien, C.-F., Deng, J.-F., 2004. A container packing support system for determining and visualizing container packing patterns. *Decision Support Systems* 37, 23-34.
- Chien, C.-F., Lee, C.-Y., Huang, Y.-C., Wu, W.-T., 2009. An efficient computational procedure for determining the container-loading pattern. *Computers & Industrial Engineering* 56, 965-978.
- Chien, C.-F., Wu, W.-T., 1998. A recursive computational procedure for container loading. *Computers & Industrial Engineering* 35, 319-322.

- Chien, C.-F., Wu, W.-T., 1999. A framework of modularized heuristics for determining the container loading patterns. *Computers & Industrial Engineering* 37, 339-342.
- Christensen, S.G., Rousøe, D.M., 2009. Container loading with multi-drop constraints. *International Transactions in Operational Research* 16, 727-743.
- Chua, C.K., Narayanan, V., Loh, J., 1998. Constraint-based spatial representation technique for the container packing problem. *Integrated Manufacturing Systems* 9, 23-33.
- Crainic, T.G., Perboli, G., Tadei, R., 2008. Extreme Point-Based Heuristics for Three-Dimensional Bin Packing. *INFORMS Journal on Computing* 20, 368-384.
- Crainic, T.G., Perboli, G., Tadei, R., 2009. TS2PACK: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research* 195, 744-760.
- Davies, A.P., Bischoff, E.E., 1999. Weight distribution considerations in container loading. *European Journal of Operational Research* 114, 509-527.
- de Almeida, A., Figueiredo, M.B., 2010. A particular approach for the Three-dimensional Packing Problem with additional constraints. *Computers & Operations Research* 37, 1968-1976.
- de Castro Silva, J., Soma, N.Y., Maculan, N., 2003. A greedy search for the three-dimensional bin packing problem: the packing static stability case. *International Transactions in Operational Research* 10, 141-153.
- Dereli, T., Das, G.S., 2011. A hybrid “bee(s) algorithm for solving container loading problems. *Applied Soft Computing* 11, 2854-2862.
- Egeblad, J., Pisinger, D., 2009. Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Computers & Operations Research* 36, 1026-1049.
- Eley, M., 2002. Solving container loading problems by block arrangement. *European Journal of Operational Research* 141, 393-409.
- Eley, M., 2003. A bottleneck assignment approach to the multiple container loading problem. *OR Spectrum* 25, 45-60.
- Elhedhli, S., Gzara, F., 2014. Characterizing the optimality gap and the optimal packings for the bin packing problem. *Optimization Letters* 9, 209-223.
- Epstein, L., Levy, M., 2010. Dynamic multi-dimensional bin packing. *Journal of Discrete Algorithms* 8, 356-372.

- Faina, L., 2000. A global optimization algorithm for the three-dimensional packing problem. *European Journal of Operational Research* 126, 340-354.
- Fanslau, T., Bortfeldt, A., 2010. A Tree Search Algorithm for Solving the Container Loading Problem. *INFORMS Journal on Computing* 22, 222-235.
- Farøe, O., Pisinger, D., Zachariasen, M., 2003. Guided Local Search for the Three-Dimensional Bin-Packing Problem. *INFORMS Journal on Computing* 15, 267-283.
- Fekete, S.P., Schepers, J., 1997. A new exact algorithm for general orthogonal d-dimensional knapsack problems, in: Burkard, R., Woeginger, G. (Eds.), *Algorithms — ESA '97, Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 144-156.
- Fekete, S.P., Schepers, J., 2004. A Combinatorial Characterization of Higher-Dimensional Orthogonal Packing. *Mathematics of Operations Research* 29, 353-368.
- Fekete, S.P., Schepers, J., Veen, J.C. van der, 2007. An Exact Algorithm for Higher-Dimensional Orthogonal Packing. *Operations Research* 55, 569-587.
- Feo, T.A., Resende, M.G.C., 1989. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters* 8, 67-71.
- Foulds, L.R., 1983. The heuristic problem-solving approach. *Journal of the Operational Research Society* 34, 927-934.
- Fujiyoshi, K., Kawai, H., Ishihara, K., 2009. A Tree Based Novel Representation for 3D-Block Packing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 759-764.
- Gehring, H., Bortfeldt, A., 1997. A Genetic Algorithm for Solving the Container Loading Problem. *International Transactions in Operational Research* 4, 401-418.
- Gehring, H., Bortfeldt, A., 2002. A Parallel Genetic Algorithm for Solving the Container Loading Problem. *International Transactions in Operational Research* 9, 497-511.
- Gehring, H., Menschner, K., Meyer, M., 1990. A computer-based heuristic for packing pooled shipment containers. *European Journal of Operational Research* 44, 277-288.
- Gendreau, M., Iori, M., Laporte, G., Martello, S., 2006. A Tabu Search Algorithm for a Routing and Container Loading Problem. *Transportation Science* 40, 342-350.
- George, J.A., 1992. A Method for Solving Container Packing for a Single Size of Box. *The Journal of the Operational Research Society* 43, 307-312.

- George, J.A., Robinson, D.F., 1980. A heuristic for packing boxes into a container. *Computers & Operations Research* 7, 147-156.
- Ghirardi, M., Potts, C.N., 2005. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research, Project Management and Scheduling* 165, 457-467.
- Glover, F., 1986. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computer & Operations Research* 13, 533-549.
- Glover, F., 1989. Tabu Search - Part I. *ORSA Journal on Computing* 1(3), 190-206.
- Glover, F., 1990. Tabu Search - Part II. *ORSA Journal on Computing* 2(1), 4-32.
- Gonçalves, J.F., Resende, M.G.C., 2012. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research* 39, 179-190.
- Gonçalves, J.F., Resende, M.G.C., 2013. A biased random key genetic algorithm for 2D and 3D bin packing problems. *International Journal of Production Economics* 145, 500-510.
- Gonzalez, Y., Miranda, G., Leon, C., 2016. Multi-objective Multi-level Filling Evolutionary Algorithm for the 3D Cutting Stock Problem. *Procedia Computer Science, Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 20th International Conference KES-2016* 96, 355-364.
- Haessler, R.W., Talbot, F.B., 1990. Load planning for shipments of low density products. *European Journal of Operational Research* 44, 289-299.
- Han, C.P., Knott, K., Egbelu, P.J., 1986. A heuristic approach to the three dimensional cargo loading problem. *Computers & Industrial Engineering* 11, 109-113.
- Hasni, H., Sabri, H., 2013. On a Hybrid Genetic Algorithm for Solving the Container Loading Problem with no Orientation Constraints. *Journal of Mathematical Modelling and Algorithms in Operations Research* 12, 67-84.
- He, K., Huang, W., 2010. A caving degree based flake arrangement approach for the container loading problem. *Computers & Industrial Engineering* 59, 344-351.
- He, K., Huang, W., 2011. An efficient placement heuristic for three-dimensional rectangular packing. *Computers & Operations Research* 38, 227-233.

- He, Y., Wu, Y., de Souza, R., 2012. A global search framework for practical three-dimensional packing with variable carton orientations. *Computers & Operations Research* 39, 2395-2414.
- Hemminki, J., 1994. Container loading with variable strategies in each layer. University of Turku, Institute for Applied Mathematics.
- Hifi, M., 2002. Approximate algorithms for the container loading problem. *International Transactions in Operational Research* 9, 747-774.
- Hifi, M., Kacem, I., N gre, S., Wu, L., 2010. A Linear Programming Approach for the Three-Dimensional Bin-Packing Problem. *Electronic Notes in Discrete Mathematics* 36, 993-1000.
- Hifi, M., Negre, S., Wu, L., 2014. Hybrid greedy heuristics based on linear programming for the three-dimensional single bin-size bin packing problem. *International Transactions in Operational Research* 21, 59-79.
- Holland, J.H., 1975. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor (reprinted by The MIT Press, 1992).
- Huang, W., He, K., 2009a. A caving degree approach for the single container loading problem. *European Journal of Operational Research* 196, 93-101.
- Huang, W., He, K., 2009b. A new heuristic algorithm for cuboids packing with no orientation constraints. *Computers & Operations Research* 36, 425-432.
- Huang, Y.-H., Hwang, F.J., Lu, H.-C., 2016. An effective placement method for the single container loading problem. *Computers & Industrial Engineering* 97, 212-221.
- Ivancic, N., Mathur, K., Mohanty, B., 1989. An integer programming based heuristic approach to the three-dimensional packing problem. *Journal of Manufacturing and Operations Management* 2, 268-298.
- Jamrus, T., Chien, C.-F., 2016. Extended priority-based hybrid genetic algorithm for the less-than-container loading problem. *Computers & Industrial Engineering* 96, 227-236.
- Jin, Z., Ito, T., Ohno, K., 2003. The Three-Dimensional Bin Packing Problem and Its Practical Algorithm. *JSME International Journal Series C Mechanical Systems, Machine Elements and Manufacturing* 46, 60-66.
- Junqueira, L., Morabito, R., Sato Yamashita, D., 2012a. Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers & Operations Research*

search 39, 74-85.

Junqueira, L., Morabito, R., Sato Yamashita, D., 2012b. MIP-based approaches for the container loading problem with multi-drop constraints. *Annals of Operations Research* 199, 51-75.

Kang, K., Moon, I., Wang, H., 2012. A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. *Applied Mathematics and Computation* 219, 1287-1299.

Kang, M.-K., Jang, C.-S., Yoon, K.-S., 2010. Heuristics with a new block strategy for the single and multiple containers loading problems. *Journal of the Operational Research Society* 61, 95-107.

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by Simulated Annealing. *Science* 220, 671-680.

Lai, K.K., Chan, J.W.M., 1997. Developing a simulated annealing algorithm for the cutting stock problem. *Computers & Industrial Engineering* 32, 115-127.

Lai, K.K., Xue, J., Xu, B., 1998. Container packing in a multi-customer delivering operation. *Computers & Industrial Engineering* 35, 323-326.

Li, X., Zhang, K., 2015. A hybrid differential evolution algorithm for multiple container loading problem with heterogeneous containers. *Computers & Industrial Engineering* 90, 305-313.

Liang, S.-C., Lee, C.-Y., Huang, S.-W., 2007. A hybrid meta-heuristic for the container loading problem. *Communications of the International Information Management Association* 7, 73-84.

Lim, A., Ma, H., Xu, J., Zhang, X., 2012. An iterated construction approach with dynamic prioritization for solving the container loading problems. *Expert Systems with Applications* 39, 4292-4305.

Lim, A., Rodrigues, B., Wang, Y., 2003. A multi-faced buildup algorithm for three-dimensional packing problems. *Omega* 31, 471-481.

Lim, A., Rodrigues, B., Yang, Y., 2005. 3-D Container Packing Heuristics. *Applied Intelligence* 22, 125-134.

Lim, A., Zhang, X., 2005. The container loading problem, in: *Proceedings of the 2005 ACM Symposium on Applied Computing, SAC '05*. ACM, New York, NY, USA, pp. 913-

917.

Lin, J.-L., Foote, B., Pulat, S., Chang, C.-H., Cheung, J.Y., 1993. Hybrid genetic algorithm for container packing in three dimensions, in: *Proceedings of the Ninth Conference on Artificial Intelligence for Applications*. IEEE Computer Society Press, pp. 353-359.

Lins, L., Lins, S., Morabito, R., 2002. An n-tet graph approach for non-guillotine packings of n-dimensional boxes into an n-container. *European Journal of Operational Research* 141, 421-439.

Liu, S., Tan, W., Xu, Z., Liu, X., 2014. A tree search algorithm for the container loading problem. *Computers & Industrial Engineering* 75, 20-30.

Liu, J., Yue, Y., Dong, Z., Maple, C., Keech, M., 2011a. A novel hybrid tabu search approach to container loading. *Computers & Operations Research* 38, 797-807.

Liu, W.-Y., Lin, C.-C., Yu, C.-S., 2011b. On the Three-Dimensional Container Packing Problem Under Home Delivery Service. *Asia-Pacific Journal of Operational Research* 28, 601-621.

Liu, J., Zhang, X., Yue, Y., Maple, C., Simos, T.E., Psihoyios, G., Tsitouras, C., Anastassi, Z., 2012. Effectively handling three-dimensional spaces for container loading. *AIP Conference Proceedings* 1479, 1960-1963.

Lodi, A., Martello, S., Vigo, D., 2002. Heuristic algorithms for the three-dimensional bin packing problem. *European Journal of Operational Research* 141, 410-420.

Lodi, A., Martello, S., Vigo, D., 2004. TSPack: A Unified Tabu Search Code for Multi-Dimensional Bin Packing Problems. *Annals of Operations Research* 131, 203-213.

Loh, T., Nee, A., 1992. A packing algorithm for hexahedral boxes, in: *Proceedings of the Conference of Industrial Automation*. Singapore, pp. 115-126.

Mack, D., Bortfeldt, A., Gehring, H., 2004. A parallel hybrid local search algorithm for the container loading problem. *International Transactions in Operational Research* 11, 511-533.

Martello, S., Pisinger, D., Vigo, D., 2000. The Three-Dimensional Bin Packing Problem. *Operations Research* 48, 256-267.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E., 1953. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics* 21, 1087-1092.

- Miyazawa, F.K., Wakabayashi, Y., 1997. An algorithm for the three-dimensional packing problem with asymptotic performance analysis. *Algorithmica* 18, 122-144.
- Miyazawa, F.K., Wakabayashi, Y., 1999. Approximation Algorithms for the Orthogonal Z-Oriented Three-Dimensional Packing Problem. *SIAM Journal on Computing* 29, 1008-1029.
- Miyazawa, F.K., Wakabayashi, Y., 2009. Three-dimensional packings with rotations. *Computers & Operations Research* 36, 2801-2815.
- Mladenović, N., Hansen, P., 1997. Variable Neighborhood Search. *Computers & Operations Research* 24, 1097-1100.
- Mohanty, B.B., Mathur, K., Ivancic, N.J., 1994. Value considerations in three-dimensional packing — A heuristic procedure using the fractional knapsack problem. *European Journal of Operational Research* 74, 143-151.
- Moon, I., Nguyen, T.V.L., 2014. Container packing problem with balance constraints. *OR Spectrum* 36, 837-878.
- Morabito, R., Arenales, M., 1994. An AND/OR-graph approach to the container loading problem. *International Transactions in Operational Research* 1, 59-73.
- Moura, A., Oliveira, J.F., 2005. A GRASP approach to the container-loading problem. *IEEE Intelligent Systems* 20, 50-57.
- Mustafee, N., Bischoff, E.E., 2013. Analysing trade-offs in container loading: combining load plan construction heuristics with agent-based simulation. *International Transactions in Operational Research* 20, 471-491.
- Ngoi, B.K.A., Tay, M.L., Chua, E.S., 1994. Applying spatial representation techniques to the container packing problem. *International Journal of Production Research* 32, 111-123.
- Osman, I.H., Kelly, J.P. (Eds.), 1996. *Meta-Heuristic: Theory and Applications*. Kluwer Academic Publishers, Boston, MA..
- Padberg, M., 2000. Packing small boxes into a big box. *Mathematical Methods of Operations Research* 52, 1-21.
- Parreño, F., Alvarez-Valdes, R., Oliveira, J., Tamarit, J., 2010. Neighborhood structures for the container loading problem: a VNS implementation. *Journal of Heuristics* 16, 1-22.

- Parreño, F., Alvarez-Valdes, R., Tamarit, J.M., Oliveira, J.F., 2008. A Maximal-Space Algorithm for the Container Loading Problem. *INFORMS Journal on Computing* 20, 412-422.
- Pisinger, D., 2002. Heuristics for the container loading problem. *European Journal of Operational Research* 141, 382-392.
- Ramos, A.G., Oliveira, J.F., Gonçalves, J.F., Lopes, M.P., 2015. Dynamic stability metrics for the container loading problem. *Transportation Research Part C: Emerging Technologies* 60, 480-497.
- Ramos, A.G., Oliveira, J.F., Gonçalves, J.F., Lopes, M.P., 2016. A container loading algorithm with static mechanical equilibrium stability constraints. *Transportation Research Part B: Methodological* 91, 565-581.
- Ratcliff, M.S.W., 1996. Aspects of Container Loading. Ph.D. thesis, European Business Management School, University of Wales Swansea, Swansea, U.K.
- Ratcliff, M.S.W., Bischoff, E.E., 1998. Allowing for weight considerations in container loading. *OR Spektrum* 20, 65-71.
- Reeves, C.R., Beasley, J.E., 1995. Introduction, in: Reeves, C.R. (Eds.), *Modern Heuristic Techniques for Combinatorial problems*. Advance Topics in Computer Science, pp. 1-19.
- Ren, J., Tian, Y., Sawaragi, T., 2011. A tree search method for the container loading problem with shipment priority. *European Journal of Operational Research* 214, 526-535.
- Resende, M.G.C., Ribeiro, C.C., 2003. Greedy randomized adaptive search procedures, in: Glover, F., Kochenberger, G. (Eds.), *Handbook of Metaheuristics 2003*, Kluwer Academic Publishers, pp. 219-249.
- Sabuncuoglu, I., Bayiz, M., 1999. Job shop scheduling with beam search. *European Journal of Operational Research* 118, 390-412.
- Scheithauer, G., 1991. A three-dimensional bin packing algorithm. *Journal of Information Processing and Cybernetics* 27, 263-271.
- Scheithauer, G., 1992. Algorithms for the Container Loading Problem, in: Gaul, W., Bachem, A., Habenicht, W. (Eds.). Presented at the Operations Research Proceedings 1991, Springer Berlin Heidelberg, pp. 445-452.
- Sciomachen, A., Tanfani, E., 2007. A 3D-BPP approach for optimising stowage plans and

- terminal productivity. *European Journal of Operational Research* 183, 1433-1446.
- Silveira, M.E., Vieira, S.M., Sousa, J.M.D.C., 2013. An ACO Algorithm for the 3D Bin Packing Problem in the Steel Industry, in: Ali, M., Bosse, T., Hindriks, K.V., Hoogendoorn, M., Jonker, C.M., Treur, J. (Eds.), *Recent Trends in Applied Artificial Intelligence, Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 535-544.
- Soak, S.-M., Lee, S.-W., Yeo, G.-T., Jeon, M.-G., 2008. An effective evolutionary algorithm for the multiple container packing problem. *Progress in Natural Science* 18, 337-344.
- Takahara, S., 2006. A Simple Meta-heuristic Approach for the Multiple Container Loading Problem, in: *IEEE International Conference on Systems, Man and Cybernetics, 2006. SMC '06*. pp. 2328-2333.
- Takahara, S., 2008. A Multi-start Local Search Approach to the Multiple Container Loading Problem, in: Bednorz, W. (Ed.), *Advances in Greedy Algorithms*. IN-TECH, pp. 55-68.
- Thapatsuwan, P., Pongcharoen, P., Hicks, C., Chainate, W., 2012. Development of a stochastic optimisation tool for solving the multiple container packing problems. *International Journal of Production Economics* 140, 737-748.
- Tian, T., Zhu, W., Lim, A., Wei, L., 2016. The multiple container loading problem with preference. *European Journal of Operational Research* 248, 84-94.
- Trivella, A., Pisinger, D., 2016. The load-balanced multi-dimensional bin-packing problem. *Computers & Operations Research* 74, 152-164.
- Voudouris, C., 1997. Guided local search for combinatorial optimisation problems. Ph.D. thesis, Department of Computer Science, University of Essex, Colchester, U.K.
- Voudouris, C., Tsang, E., 1999. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research* 113, 469-499.
- Wang, Z., Li, K.W., Levy, J.K., 2008. A heuristic for the container loading problem: A tertiary-tree-based dynamic space decomposition approach. *European Journal of Operational Research* 191, 86-99.
- Wang, N., Lim, A., Zhu, W., 2013. A multi-round partial beam search approach for the single container loading problem with shipment priority. *International Journal of Production Economics* 145, 531-540.
- Wäscher, G., Haußner, H., Schumann, H., 2007. An improved typology of cutting and

- packing problems. *European Journal of Operational Research* 183, 1109-1130.
- Wei, L., Zhu, W., Lim, A., 2015. A goal-driven prototype column generation strategy for the multiple container loading cost minimization problem. *European Journal of Operational Research* 241, 39-49.
- Wu, Y., Li, W., Goh, M., de Souza, R., 2010. Three-dimensional bin packing problem with variable bin height. *European Journal of Operational Research* 202, 347-355.
- Xue, J., Lai, K.K., 1997. Effective methods for a container packing operation. *Mathematical and Computer Modelling* 25, 75-84.
- Yeh, J.M., Lin, Y.C., Yi, S., 2003. Applying genetic algorithms and neural networks to the container loading problem. *Journal of Information and Optimization Sciences* 24, 423-443.
- Yeung, L.H.W., Tang, W.K.S., 2005. A hybrid genetic approach for container loading in logistics industry. *IEEE Transactions on Industrial Electronics* 52, 617-627.
- Zhang, D., Peng, Y., Leung, S.C.H., 2012. A heuristic block-loading algorithm based on multi-layer search for the container loading problem. *Computers & Operations Research* 39, 2267-2276.
- Zheng, J.-N., Chien, C.-F., Gen, M., 2015. Multi-objective multi-population biased random-key genetic algorithm for the 3-D container loading problem. *Computers & Industrial Engineering, Maritime logistics and transportation intelligence* 89, 80-87.
- Zhu, W., Huang, W., Lim, A., 2012a. A prototype column generation strategy for the multiple container loading problem. *European Journal of Operational Research* 223, 27-39.
- Zhu, W., Lim, A., 2012. A new iterative-doubling Greedy-Lookahead algorithm for the single container loading problem. *European Journal of Operational Research* 222, 408-417.
- Zhu, W., Oon, W.-C., Lim, A., Weng, Y., 2012b. The six elements to block-building approaches for the single container loading problem. *Applied Intelligence* 37, 431-445.
- Zhu, W., Zhang, Z., Oon, W.-C., Lim, A., 2012c. Space defragmentation for packing problems. *European Journal of Operational Research* 222, 452-463.

