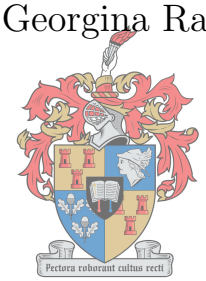
Metaheuristic solution of the two-dimensional strip packing problem

Rosephine Georgina Rakotonirainy



UNIVERSITEIT
iYUNIVESITHI
STELLENBOSCH
UNIVERSITY

Dissertation presented for the degree of
Doctor of Philosophy
in the Faculty of Engineering at Stellenbosch University

Promoter: Prof JH van Vuuren

December 2018

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2018

Abstract

In the two-dimensional strip packing problem, the objective is to pack a set of rectangular items in a non-overlapping manner into a single, rectangular object of fixed width but unlimited height, such that the resulting height of the packed items is a minimum. This problem has a wide range of applications, especially in the wood, glass and paper industries. Over the past few years, the development of fast and effective packing algorithms — mainly employing heuristic and metaheuristic techniques — has been the major concern of most strip packing-related research due to the complexity and combinatorial nature of the problem.

A new systematic way of comparing the relative performances of strip packing algorithms is introduced in this dissertation. A large, representative set of strip packing benchmark instances from various repositories in the literature is clustered into different classes of test problems based on their underlying features. The various strip packing algorithms considered are all implemented on the same computer, and their relative effectiveness is contrasted for the different data categories. More specifically, the aim in this dissertation is to study the effect of characteristics inherent to the benchmark instances employed in comparisons of the relative performances of various strip packing algorithms, with a specific view to develop decision support capable of identifying the most suitable algorithms for use in the context of specific classes of strip packing problem instances.

Two improved strip packing metaheuristics are also proposed in this dissertation. These algorithms have been designed in such a way as to improve on the effectiveness of existing algorithms. The two newly proposed algorithms are compared with a representative sample of metaheuristics from the literature in terms of both solution quality achieved and execution time required in the context of the clustered benchmark data. It is found that the new algorithms indeed compare favourably with other existing strip packing metaheuristics in the literature. It is also found that specific properties of the test problems affect the solution qualities and relative rankings achieved by the various packing algorithms.

One of the most important findings in this dissertation is that the characteristics of the benchmark instances considered for comparative algorithmic study purposes should be taken into account in the future in order to avoid biased research conclusions.

Uittreksel

In die twee-dimensionele strookinpakkingsprobleem moet 'n versameling reghoekige voorwerpe op 'n nie-oorvleuelende wyse in 'n enkele reghoekige voorwerp van vaste breedte, maar onbeperkte hoogte, gepak word sodat die resulterende hoogte van die ingepakte voorwerpe 'n minimum is. Hierdie probleem het 'n wye verskeidenheid toepassings, veral in die hout-, glas- en papierbedrywe. Oor die afgelope paar jaar het die ontwikkeling van vinnige en doeltreffende inpakkingsalgoritmes — wat hoofsaaklik berus op heuristiese en metaheuristiese tegnieke — aanleiding gegee tot die meeste strookinpakkings-verwante navorsing weens die kompleksiteit en kombinatoriese aard van die probleem.

'n Nuwe sistematiese manier word in hierdie proefskrif daargestel vir die relatiewe vergelyking van die doeltreffendheid van strookinpakkingsalgoritmes. 'n Groot, verteenwoordigende versameling strookinpakkingstoetsprobleme uit verskillende versamelings in die literatuur word in 'n aantal klasse toetsprobleme op grond van hul onderliggende kenmerke gegroepeer. Die onderskeie strookinpakkingsalgoritmes wat oorweeg word, word almal op dieselfde rekenaar geïmplementeer, en hul relatiewe doeltreffendhede word in die konteks van hierdie verskillende datakategorieë vergelyk. Die doel van hierdie proefskrif is om die effek van eienskappe onderliggend aan die toetsprobleme wat tydens die relatiewe vergelyking van verskeie strookinpakkingsalgoritmes ingespan word, te bestudeer, met die oog op die ontwikkeling van besluitsteun ten opsigte van die mees gepaste algoritmes vir gebruik in die konteks van spesifieke klasse strookinpakkingsprobleemgevalle.

Twee verbeterde strookinpakkingsmetaheuristieke word ook in hierdie proefskrif voorgestel. Hierdie algoritmes is ontwerp om op die doeltreffendheid van bestaande algoritmes te verbeter. Die twee nuwe algoritmes word met 'n verteenwoordigende steekproef van metaheuristieke uit die literatuur ten opsigte van beide oplossingskwaliteit behaal en uitvoeringstyd vereis, in die konteks van die gegroepeerde toetsdata, vergelyk. Daar word bevind dat die nuwe algoritmes inderdaad gunstig vergelyk met ander bestaande strookinpakkingsmetaheuristieke in die literatuur. Daar word ook bevind dat spesifieke eienskappe van die toetsdata die oplossingskwaliteit en relatiewe prestasie van die verskillende algoritmes beïnvloed.

Een van die belangrikste bevindings in hierdie proefskrif is dat daar in die toekoms rekening gehou moet word met die eienskappe van toetsprobleme wat vir vergelykende algoritmiese studiedoeleindes ingespan word om sodoende bevooroordeelde navorsingsgevolgtrekkings te vermy.

Acknowledgements

The author wishes to acknowledge the following people and institutions for their various contributions towards the completion of this work:

- My supervisor, Prof JH van Vuuren, for his dedication and guidance throughout the completion of this dissertation. I am very grateful for his patience and support. Thank you, Prof, for taking me on as your student and for welcoming me to the *Stellenbosch Unit for Operations Research in Engineering* (SUnORE) research group. I feel privileged to have been part of the group and to have had a supervisor who gives his time to ensure that the work contained in this research is of a high quality.
- My fellow SUnORE members for their friendship and for making the past three years an unforgettable time. The experiences, all the interesting social activities, and other memories will be cherished.
- Prof Martin Kidd, for the help with the statistical analysis of the data.
- Almost-Dr Thorsten Schmidt-Dumont, for helping me with some drawings in Ipe.
- The Industrial Engineering Department at Stellenbosch University, as well as SUnORE, for the use of its office space and computing facilities.
- The *Deutscher Akademischer Austauschdienst* (DAAD) German Academic Exchange Service with *African Institute for Mathematical Sciences* (AIMS) for the financial support received over the past three years.
- My family and other friends for their encouragement and moral support.

"So do not fear, for I am with you; do not be dismayed, for I am your God. I will strengthen you and help you; I will uphold you with my righteous right hand." Isaiah 41:10

Thanks be to God for His mercy

Table of Contents

List of Acronyms

ANOVA: ANalysis Of VAriance

BF: Best-Fit

BFDH*: Modified Best-Fit Decreasing Height

BL: Bottom-Left

CH: Constructive Heuristic

CLP-GA : Genetic Algorithm for the Container Loading Problem

CLP-SA : Simulated Annealing for the Container Loading Problem

CX: Cycle Crossover

C&P: Cutting and Packing

DA: Decreasing Area

DH: Decreasing Height

DP: Decreasing Perimeter

DW: Decreasing Width

DBSCAN: Density-Based Spatial Clustering of Applications with Noise

GA: Genetic Algorithm

GRASP: Greedy Randomised Adaptive Search Procedure

HGA: Hybrid Genetic Algorithm

HSA: Hybrid Simulated Annealing

IA: Improved Algorithm

IAm: Modified Improved Algorithm

IHR: Improved Heuristic Recursive

ISA: Intelligent Search Algorithm

PCA: Principal Component Analysis

PMX: Partially Matched Crossover

POS: POsition-based Crossover

SA: Simulated Annealing

SPGAL: Strip Packing Genetic Algorithm Layer approach

SPSAL: Strip Packing Simulated Annealing Layer approach

SPP: Strip Packing Problem

SRA: Simple Randomised Algorithm

SUS: Stochastic Universal Selection

2D SPP: Two-Dimensional Strip Packing Problem

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Contents

Cutting and packing (C&P) problems are complex combinatorial optimisation problems with a wide variety of applications. The variety of the different problems in this class is as large as their application areas, spanning disciplines such as the management sciences [72], mathematics [45], computer science [38], and operations research [55]. Due to this diversity of application area, the research field of C&P problems has received significant attention in the literature. These problems have been researched extensively, especially in the operations research literature, since 1939 [104]. The continued interest in this field is, in part, a result of the need to develop automated packing layouts and cutting patterns for industry so as to achieve a more effective utilisation of resources than is intuitively possible.

This dissertation is a study of a specific type of C&P problem. Details of the problem considered, as well as the main objectives of the study, are outlined in this chapter. A brief general background on C&P problems is provided in §1.1, while a more thorough overview of the literature on these problems, and the scope of the dissertation, follow in the next chapter. The problem under investigation in this dissertation is made more precise in §1.2. This is followed by a presentation of the dissertation aim in §1.3 and the dissertation objectives pursued in §1.4. A general preview of the dissertation organisation is finally provided in §1.5.

1.1 Background

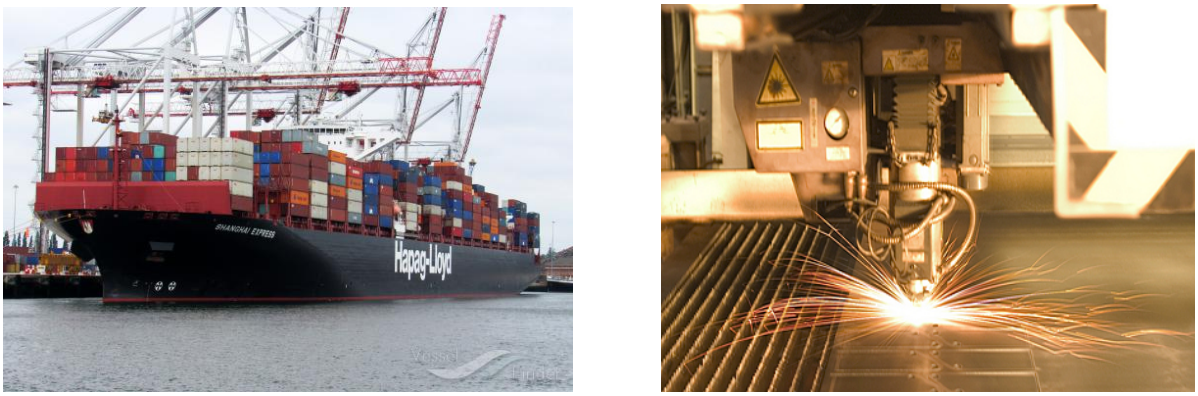
C&P problems, in general, consist of partitioning large items into smaller pieces of specified dimensions in such a manner that waste is minimised, or of arranging smaller pieces of specified dimensions into larger objects in a non-overlapping manner, again minimising wasted area. Several names have been proposed in the literature for this class of problems, such as *trim loss and assortment problems* [85], *bin packing problems* [41], *container loading problems* [134] and *nesting problems* [56]. All these problems share a common logical structure, and the C&P problem solution process produces a layout obtained from geometric combinations of smaller

items assigned to larger objects. The waste resulting from this assignment, also referred to *residual space* or *trim loss*, has to be minimised in each case.

C&P problems arise in many real-world applications, ranging from logistics (*e.g.* box packing) to industrial design (*e.g.* garment manufacturing). These problems may be classified according to their various attributes, including dimensionality, the shapes of the small items, the assortment of the large objects, the particular problem constraints (*e.g.* rotatability, guillotinability) and the packing or cutting objective.

In many applications, the problem is usually considered in two-dimensional or in three-dimensional space (although such problems may also be considered in one-dimensional space [58, 64, 100] or in higher-dimensional space [13, 38, 132]). An example instance of the two-dimensional case consists of cutting a stock of paper into smaller parts, while an example instance of the three-dimensional variant involves the loading of goods into containers (as illustrated in Figure 1.1(a)). The majority of the existing literature on C&P problems pertains to the case where the items to be packed are regular in nature [25, 31, 91, 165]. Some studies have, however, also been conducted in the context of packings involving irregular shapes, which sometimes manifests itself in the apparel industry [19, 20, 56, 71].

In respect of the assortment of the large objects, the problem may consist of packing items into a large object of fixed width and variable height with the objective of minimising the packing height, known as the *strip packing problem* (SPP) which finds application in the metal industry as illustrated in Figure 1.1(b), or into a set of bins of fixed width and height with the objective of minimising the number of bins utilised, referred to as the *bin packing problem*, as in box packing. A set of packing constraints usually has to be satisfied, such as, when cutting items out of wooden planks exhibiting grain patterns, where rotation of the items cut is usually disallowed. In the glass industry, a so-called guillotine-cut constraint is often applied whereby a series of cuts right through the large object parallel or perpendicular to its edges is required.



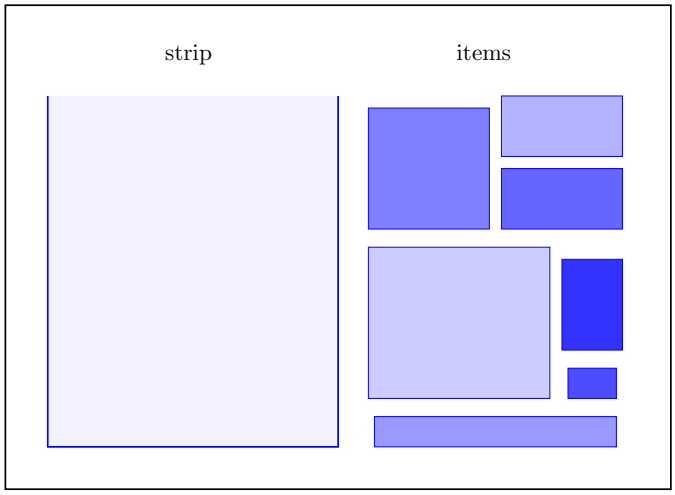
(a) *A container vessel [154]* (b) *Cutting steel plate [2]*

FIGURE 1.1: *Examples of cutting and packing problems in real-world applications.*

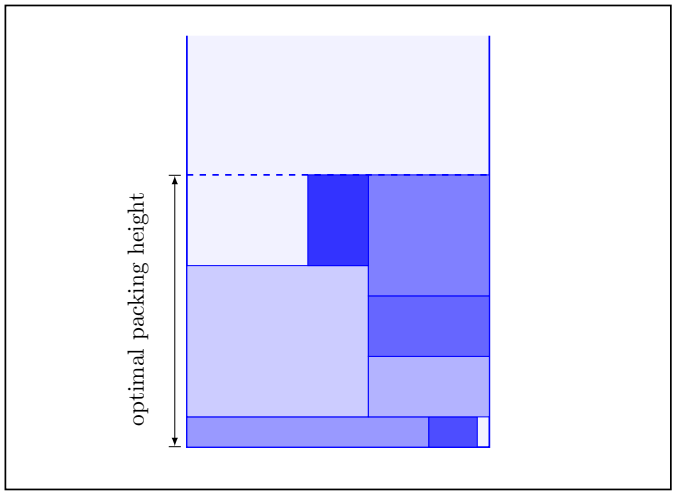
1.2 Informal Problem Description

This dissertation is a study of a very specific two-dimensional rectangular packing problem, called the *two-dimensional strip packing problem* (2D SPP). As mentioned in the previous section, this problem consists of packing a set of rectangular items into a large rectangular object of fixed width and unlimited height (referred to as a *strip*) in such a manner that the resulting height

of the packed items is a minimum. An illustrative example of a 2D SPP instance is provided in Figure 1.2. A significant challenge in the C&P literature is to design suitable packing strategies according to which optimal or near-optimal packing layouts can be generated for (Figure 1.2(b)) of a given set of items to be packed into the given strip (as illustrated in Figure 1.2(a)) within reasonable time frames. Savić *et al.* [140] claimed that "a successful optimal solution or even finding an approximately good solution significantly facilitates in both saving money and raw materials."



(a) *Initial packing objects*



(b) *Packed items*

FIGURE 1.2: *An example instance of the 2D SPP.*

1.3 Dissertation Aim

Due to its important industrial and commercial applications, the 2D SPP has received significant attention in the operations research literature during the last six decades. The development of efficient and effective packing algorithms has been the major concern of many researchers due to the complexity and combinatorial nature of the problem [37, 93, 113, 120, 121, 128, 131]. Not only should the solution quality be considered (as dictated in industrial applications), but also

the computational cost of the packing method. *Heuristic* and *metaheuristic techniques* achieve suitable trade-offs between achieving these requirements, and so they are the favoured choices in terms of packing methodology in most practical applications.

The aim in this dissertation is to contribute to the class of metaheuristic approaches toward solving the 2D SPP. Different classes of heuristics and metaheuristics have been applied successfully to this class of C&P problems [11, 25, 29, 91, 99, 116, 118, 164]. Most of the recently proposed solution approaches are based on hybrid approaches in which heuristic and metaheuristic techniques are combined in order to achieve good packing solution quality over the entire range of existing benchmark instances available in the literature. Numerical results indicate that metaheuristics, as well as these hybrid techniques, outperform heuristic packing routines by a large margin in terms of solution quality in general. Nevertheless, some of these algorithms require significant computation times to find good solutions for large-scale problem instances.

Furthermore, no researcher in the field of C&P problems has compared the relative performances of these metaheuristic algorithms in the context of a single, large set of packing problem instances with various characteristics, on the same platform. Hopper and Turton [91] have carried out significant research in this area, but they only compared three hybrid metaheuristic algorithms in respect of small data sets involving seven different categories of problem instances generated by themselves. More comprehensive, overarching numerical tests on the same hardware platform are, however, of considerable interest when comparing the effectiveness of metaheuristic algorithms based on a variety of problem instance characteristics in an unbiased fashion.

Based on a novel classification of a large set of benchmark instances from the literature according to different characteristics, the relative performances of a number of metaheuristic solution techniques for the 2D SPP are compared in this study, with the aim of developing decision support capable of recommending the most effective algorithms for use in the context of various classes of benchmark instances. More specifically, the aim in this dissertation is twofold. The first aim is to identify a large, representative set of suitable benchmark instances for the 2D SPP from various repositories in the literature and to classify them into different classes of test problems based on their underlying features. More importantly, the second aim is to propose improved 2D SPP metaheuristics and to compare their relative performances with those of existing 2D SPP metaheuristics in respect of the clustered benchmark instances obtained in pursuit of the first aim. The main result of the dissertation is a characterisation of the effectiveness of the various algorithms under investigation in respect of their appropriateness of application to the clustered benchmark data under different algorithmic execution time budgets.

1.4 Dissertation Objectives

The above-mentioned aims are accomplished by pursuing the following ten objectives:

I To *conduct* a literature survey with respect to C&P problems in general in order to delimit the scope of the study in a sensible fashion. This includes literature on

  (a) the classification of existing types of C&P problems in the literature, and

  (b) studies of the theoretical and practical aspects of such problems.

II To *perform* a general literature study with respect to methods typically employed to solve C&P problems. This includes the following three classes of techniques:

  (a) exact solution approaches,

(b) heuristic solution techniques, and

(c) metaheuristic solution techniques.

III To *perform* a more specific literature study with respect to existing algorithmic approaches for the 2D SPP. This study encompasses the literature related to

(a) known strip packing heuristics, and

(b) recently proposed strip packing metaheuristics.

IV To *identify* a large set of benchmark instances of the 2D SPP in terms of which the qualities of solutions produced by all the algorithms considered in Objective III may be compared.

V To *classify* the set of benchmark instances identified in pursuit of Objective IV into different classes of test problems based on a set of features which best describe these instances. This objective is achieved by conducting a relevant clustering analysis in respect of the benchmark data collected in fulfilment of Objective IV.

VI To *implement* a representative class of the heuristic and metaheuristic algorithms reviewed in pursuit of Objective III for the 2D SPP on a personal computer, and to *apply* them to the clustered benchmark data obtained in fulfilment of Objective V. This step is aimed at identifying the strengths and weaknesses of the various algorithms in respect of SPP instances with various input data characteristics.

VII To *propose* new metaheuristics for the 2D SPP that improve on the performance of the existing methods implemented in pursuit of Objective VI. This involves

(a) the proposal of modifications to the existing metaheuristic algorithms, and

(b) the identification of superior implementations of these algorithms.

VIII To *implement* on a personal computer the improved algorithms designed in fulfilment of Objective VII.

IX To *perform* in a statistically justifiable manner an appraisal of all algorithmic approaches under investigation in terms of the solution qualities they yield and their execution times, in the context of the clustered strip packing benchmark instances of Objective V. This includes

(a) an appraisal of existing strip packing heuristics reviewed in pursuit of Objective III(a),

(b) the pursuit of efficient implementations of known (hybrid) metaheuristics identified in fulfilment of Objective III(b), and

(c) an appraisal of all strip packing metaheuristics under consideration (including the existing metaheuristics of Objective III(b) and the newly proposed algorithmic improvements of Objective VII).

X To *perform* an appraisal of the contributions made during the study, and to *recommend* follow-up work which may be pursued in future.

1.5 Dissertation Organisation

This dissertation comprises thirteen further chapters in addition to the current introductory chapter, which are grouped into five parts. The first three-chapter part is dedicated to a review of the literature on C&P problems in general as well as on existing algorithmic approaches toward solving instances of the 2D SPP. The second part, which also consists of three chapters, is devoted to the collection, documentation, and clustering of the SPP data available in the literature. The next part comprises two chapters which contain newly proposed algorithms and their related computational studies. The fourth part contains three chapters and is dedicated to an evaluation of the relative effectiveness of the various SPP algorithmic approaches considered. The final part contains two chapters and is devoted to a summary and appraisal of the contributions of the dissertation, as well as an identification of suitable avenues of follow-up investigation.

The scope of the problem considered in this dissertation is discussed in some detail in Chapter 2. The chapter opens with an introduction to various classifications of C&P problems. This includes a description of the most prominent typologies for C&P problems. This is followed by a brief review of solution methodologies for C&P problems. The three prevailing classes of packing solution approaches, namely the class of *exact methods*, *heuristic approaches*, and *metaheuristic techniques*, are described in some detail. The type of C&P problems and the class of solution methodologies considered in the remainder of this dissertation are finally outlined.

Chapter 3 is a literature review on strip packing heuristics in which five state-of-the-art algorithms from the literature are described in detail. The first algorithm is a pseudo-level packing algorithm due to Bortfeldt [25], which consists of packing items into levels according to a specific rule. The second algorithm was proposed by Liu and Teng [118], and is a member of the class of bottom-left algorithms originally proposed by Baker *et al.* [11]. The third algorithm is due to Zhang *et al.* [164], and attempts to solve an SPP instance recursively. The last two algorithms, proposed by Burke *et al.* [29] and by Leung *et al.* [116], respectively, are plane algorithms which pack items anywhere in the space defined by the boundaries of the strip (without any level restriction) according to well-defined dynamic rules.

The last literature review chapter on C&P problems, Chapter 4, is devoted to strip packing metaheuristics. Seven well-known SPP metaheuristic approaches are described in some detail in this chapter. These include the genetic algorithmic approach of Bortfeldt [25], the greedy randomised adaptive search procedure of Alvarez-Valdés *et al.* [5], the two-stage approach of Leung *et al.* [116], the randomised algorithm of Yang *et al.* [162], and the three-phase approach proposed by Wei *et al.* [158]. Approaches which involve hybrid techniques are also reviewed. These are mainly a hybrid genetic algorithm and a hybrid simulated annealing solution approach.

The SPP benchmark data instances employed throughout this dissertation are presented in Chapter 5. Two classes of benchmark instances are considered. The first class consists of zero-waste problem instances for which the respective optimal solutions are known and do not contain any wasted regions (regions of the strip not occupied by items). This class of benchmark instances contains nine data sets. The second class consists of non-zero-waste instances for which optimal solutions are not known in all cases. Those for which optimal solutions are known furthermore involve some wasted regions. This second class of problem instances contains eleven data sets. The problem generators and methods employed to generate each of these benchmark instances are also outlined in the chapter.

A literature review on the subject of cluster analysis is provided in Chapter 6. This is aimed at informing the type of clustering techniques and methods of clustering validation employed during the clustering study performed on the available SPP benchmark data documented in

Chapter 5. The chapter opens with an overview of the topic of clustering. A general background and typical clustering processes are outlined briefly and this is followed by a presentation of the most prominent examples of clustering algorithms in the literature. Various clustering validation measures are also described.

Details of the cluster analysis performed on the SPP benchmark data of Chapter 5 are presented in Chapter 7. The first section of the chapter contains a description of the data categorisation. This encompasses descriptions of the features selected to describe the data. The clustering process and the clustering result assessment performed are covered in the second section. These include discussions on data preparation, the estimation of a suitable number of clusters, the method of selection of the best clustering algorithm, and an assessment of the quality of the data clusters obtained. The last section of the chapter is devoted to descriptions of the underlying characteristics of the various clusters of benchmark data.

Two improved strip packing metaheuristics are proposed in Chapter 8. Both of these approaches involve the method of simulated annealing. The first algorithm is a hybrid approach in which the method of simulated annealing is combined with a heuristic construction algorithm, whereas the second algorithm involves application of the method of simulated annealing directly in the space of completely defined packing layouts without any encoding of solutions. Detailed descriptions of the working of these algorithms are provided in the chapter.

In order to obtain the best results achievable by means of the two newly proposed algorithmic adaptations of the previous chapter, a suitable combination of the integrated simulated annealing algorithmic parameter values has to be selected in each case. This requires evaluation of different combinations of the algorithmic parameter values according to an experimental design. Descriptions of the evaluation study performed, as well as the results obtained, are provided in Chapter 9. In the first section of the chapter, descriptions are provided of the performance evaluation measures utilised and the statistical analysis tools applied. This is followed by a discussion on the specific implementation of the method of simulated annealing employed in the two adapted algorithms. Thereafter, the experimental design followed is presented. The computational results are finally reported.

In Chapter 10, the relative effectiveness of the five heuristic algorithms reviewed in Chapter 3 are compared in respect of the clustered benchmark instances of Chapter 7. All the results are interpreted and presented in the form of boxplots and tables of *post hoc* statistical test results. The comparison is carried out at a 95% level of confidence.

Chapter 11 is devoted to descriptions of a limited computational study of the implementations of the existing hybrid metaheuristics described in Chapter 4. The focus here is to identify superior implementations of the genetic algorithm and the method of simulated annealing employed in the hybrid algorithms in terms of their constituent elements (operators and parameter values). Various parameter settings of each metaheuristic algorithm are evaluated and tested for this purpose. The experimental design followed for this purpose, together with the underlying computational results, are reported in the chapter.

The relative effectiveness of all the metaheuristic solution approaches (old and new) considered in this dissertation are finally compared in respect of the clustered benchmark instances in Chapter 12. These solution approaches include the seven metaheuristics reviewed in Chapter 4 and the two newly proposed algorithmic adaptations of Chapter 8. The comparative study of the nine algorithms is again carried out at a 95% level of confidence in terms of solution quality, and the various algorithmic execution times are also noted. A characterisation of the effectiveness of these algorithms in respect of large SPP instances in each of the benchmark clusters of Chapter 7 is also provided.

Chapter 13 contains a summary of the work presented in this dissertation as well as an appraisal of the contributions made.

Chapter 14 is the final chapter of the dissertation, and contains a number of suggestions for future follow-up work.

Part I

Cutting and Packing Problems: A review

CHAPTER 2

Overview of C&P Problems

Contents

This chapter contains an overview of the literature relevant to C&P problems and is aimed at placing the topic of this dissertation in context. A detailed classification of C&P problems is provided in §2.1 for this purpose. Four known typologies of C&P problems are reviewed and summarised in §2.1.1. This is followed by descriptions of different types of C&P problems (§2.1.2). The three major solution approaches adopted in the literature for solving C&P problems are discussed thereafter, namely exact methods (§2.2.1), heuristic techniques (§2.2.2) and metaheuristic techniques (§2.2.3). The scope of the dissertation is then outlined in §2.3, and a brief summary of the contents of the chapter is finally provided in §2.4.

2.1 Classifications of C&P Problems

In order to describe the type of C&P problem considered in this dissertation, four known typologies of C&P problems are reviewed in this section. This is followed by a presentation of the main types of C&P problems in the operations research literature.

2.1.1 C&P Problem Typologies

The early literature on C&P problems was devoted to cutting stock problems (partitioning large items into small pieces while minimising waste), with Kantorovich [104] proposing the first mathematical formulation of the cutting stock problem in 1960, Eisemann [58] considering the trim loss problem in 1957, and Gilmore and Gomory [64, 65, 66] working on the cutting

stock problem during the 1960s. Later, during the 1970s and 1980s, many authors published work on other types of C&P problems, such as Johnson [100], who pursued research on the bin packing problem in 1974, and Dowsland [53], who studied two- and three-dimensional bin packing problems in 1984. Since then, C&P problems and their applications have evolved into a very active field of study, resulting in a large volume of research dealing with various aspects of these problems.

In 1990, Dyckhoff [55] attempted to unify C&P problem research in the literature by classifying the wide range of C&P problems into clearly-defined categories. He proposed a typology based on four characteristics, namely the problem dimensionality, the kind of assignment, the assortment of the large objects, and the assortment of the small items. He subsequently identified ninety six possible types of C&P problems accordingly. The dimensionality characteristic indicates whether the problem occurs in one-, two-, three- or multi-dimensional space. An example instance of a four-dimensional problem is the packing of boxes into a container within a fixed amount of time[1]. A special case of the multi-dimensional packing problem is the vector scheduling problem in which a set of jobs, each requiring different resources (*e.g.* time and memory requirements), is assigned to a fixed number of machines such that the maximum resource usage over all resources and all machines is minimised [13, 38, 132]. The kind of assignment characteristic determines whether a selection of small items must be assigned to all large objects or whether all small items are to be assigned to a selection of objects. Dyckhoff differentiated between three types of assortments of the large objects available, namely only one large object, a number of large objects of the same shape, or large objects of different shapes. Similarly, a distinction is made between four types of assortments of the small items that have to be cut or packed, namely congruent shapes, few items of different shapes, many items of few different shapes and many items of many different shapes.

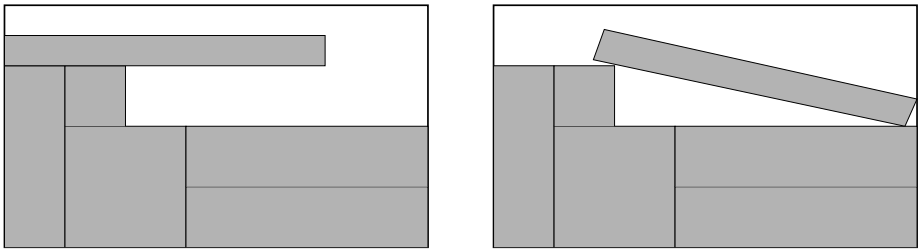
Wäscher *et al.* [157] further improved Dyckhoff's typology by making some changes and adding new components to the categories. They agreed with Dyckhoff's dimensionality characterisation and left it unchanged. They, however, proposed a different kind of assignment characteristic, namely input minimisation (a selection of small items is used to produce patterns to a set of large objects, such that all large objects are utilised), and output maximisation (a set of small items is assigned to a selection of large objects, such that all small items are considered). The assortment of small items characterisation was reduced to three types, namely a strongly homogeneous assortment of small items (all items are identical), a weakly heterogeneous assortment of small items (many items are identical), and a strongly heterogeneous assortment of small items (very few items are identical). The major change occurs in the assortment of large objects. Here Wäscher *et al.* proposed two categories, each with subcategories. The first category is the class of problems dealing with only one large object, which is partitioned into problems in which all the dimensions of the objects are fixed, those in which one dimension of the object is variable, and those in which multiple dimensions of the object are variable. The second set of problems are those dealing with several large objects. This class of problems may be partitioned into three subclasses, namely those problems in which the large objects are strongly homogeneous, those in which the objects are weakly heterogeneous and those in which the objects are strongly heterogeneous.

Lodi *et al.* [120] proposed their own subtypology for bin and strip packing problems in 1999. They concentrated on the dimensions of the problem (similar to those proposed by Dyckhoff [55] and Wäscher *et al.* [157]), the packing type (bin packing or strip packing) and the packing constraints (whether rotation is allowed and whether guillotine-cut constraints are imposed).

[1] Time is the fourth dimension in this case.

Ntene [128] also proposed a subtypology for packing problems. Her classification consists of six characteristics: the dimensionality of the problem (one-, two-, three- or higher), the shapes of the small items (regular or irregular), the assortment of the large objects (strip packing problem, single-sized bin packing problem, variable-sized bin packing problem and single bin packing problem), the nature of the information known about the items to be packed (offline if the entire list of items is known before the packing process commences, almost online if some information is known about the items before packing begins, and online if there is no prior knowledge about the list of items to be packed), the objective of the packing (minimising the strip height, minimising the number of bins used, minimising the packed area, minimising the cost of the packing, or maximising the number of items to be packed), and the set of constraints required (rotation, restriction on the placement of items, modification of the shapes of items and whether or not guillotine-cut constraints are required). A unified summary of the aforementioned C&P problem typologies is presented in Table 2.1.

In the case of regular items, the packings in all problem instances of the typologies stated above are assumed to be orthogonal, *i.e.* the packing layout exhibits patterns in which the edges of the small items are parallel or perpendicular to the edges of the large object. Orthogonal and non-orthogonal packings of regular items are illustrated in Figure 2.1.



(a) *An orthogonal packing* (b) *A non-orthogonal packing*

FIGURE 2.1: *Orthogonal and non-orthogonal packings.*

2.1.2 Types of C&P Problems

C&P problems occur in various application areas involving different constraints and objectives. In this section, the best-known basic types of C&P problems are described briefly. Some applications call for the solution of combinations of two or more of these basic types of problems. Some of the types of problems might also occur as subproblems of the others in other areas of application.

Cutting stock problems. Given an ordered list of items of specified dimensions, the problem consists of cutting the items from a given set of stock sheets such that the total cost of the stock needed to fulfill the order is minimised. This type of problem can be partitioned into two subproblems, the ***assortment problem***, which is concerned with the determination of the number and dimensions of the sheets to keep in stock, and the ***trim loss problem***, which involves the determination of the cutting pattern required to minimise waste.

Knapsack problems. Given a list of items, each associated with a value and a stock cost, the problem consists of packing the items into a fixed stock object such that the total value of the packed items is maximised.

Bin packing problems. Given a list of items of various sizes and a set of identical or differently sized bins, all items have to be packed into a minimum number of bins, in a non-overlapping

	Dyckhoff (1990)	Lodi *et al.* (1999)	Wäscher *et al.* (2006)	Ntene (2007)
Dimensionality	1 – One-dimensional 2 – Two-dimensional 3 – Three-dimensional N – N-dimensional	1 – One-dimensional 2 – Two-dimensional 3 – Three-dimensional N – N-dimensional	1 – One-dimensional 2 – Two-dimensional 3 – Three-dimensional N – N-dimensional	1 – One-dimensional 2 – Two-dimensional 3 – Three-dimensional HoD – Higher dimensional
Objective	B – Chosen items, all objects V – all items, chosen objects		OM – Output maximisation IM – Input minimisation	MaI – Maximise items packed MiA – Minimise packing area MiB – Minimise number of bins MiC – Minimise cost of packing MiS – Minimise strip height
Assortment of large objects	O – One large object I – Identical figures D – Different figures	SP – strip packing BP – Bin packing	O – One object Oa – all dimensions fixed Oo – one variable dimension Om – more variable dimensions Sf – Several large objects Si – strongly homogeneous Sw – weakly heterogeneous Ss – strongly heterogeneous	SP – Strip packing SB – Single bin packing MFB – Many bins (fixed size) MVB – Many bins (many sizes)
Assortment of small items	C – Congruent figures R – Many items (few sizes) M – Many items (many sizes) F – Few items (many sizes)		IS – Strongly homogeneous W – Weakly heterogeneous S – Strongly heterogeneous	R – Regular items I – Irregular
Packing type				Off – Offline Aon – Almost online On – Online
Constraints		Orientation O – oriented R – rotated by 90° Guillotine F – no guillotine G – guillotine applies		Orientation $\tau_0 = 0$: no rotation $\tau_0 = 1$: allow rotation Guillotine $\tau_g = 0$: no guillotine $\tau_g = 1$: guillotine applies Placement $\tau_p = 0$: no restriction $\tau_p = 1$: with restriction Modification $\tau_m = 0$: no modification $\tau_m = 1$: with modification

TABLE 2.1: *A unified summary of different typologies for packing problems available in the literature.*

manner. A variant of this problem is the ***single bin packing problem*** (in two dimensions) or ***container loading problem*** (in three dimensions), where as many small items as possible are to be packed into a single bin in order to maximise the space or volume utilisation.

Strip packing problems. A list of items has to be packed into a strip with one unlimited dimension and the goal is to minimise the packing height in the two-dimensional strip packing problem or the container length in the three-dimensional strip packing problem.

2.2 C&P Solution Methodologies

Due to their practical applicability, much has been written on the C&P problems outlined above. The theoretically oriented research has mainly focused on worst-case performance analyses of approximation algorithms [42, 43, 44]. An example is the use of a performance measure (*e.g.* an asymptotic performance bound) which enables a comparison of the optimal solution to a problem instance with the solution obtained by an algorithm. The practically and computationally oriented research, on the other hand, has concentrated on the design of solution approaches to solve instances of the various C&P problems. Various approaches have been proposed in the literature for solving C&P packing problems. These approaches may be classified into the classes of *exact methods*, *heuristic approaches* and *metaheuristic techniques*. Exact methods are typically based on a mathematical programming modelling approach and find a best packing solution, but are slow and may hence only be used to solve small problem instances. Heuristic and metaheuritsic techniques, on the other hand, are approximate solution approaches that attempt to provide near-optimal solutions in minimal time. They are more practical and provide solutions to large problem instances within reasonable time frames. A brief review of these solution strategies for C&P problems is provided in this section.

2.2.1 Exact C&P Solution Approaches

Exact packing methods, also called *deterministic packing methods*, are typically based on a mathematical programming modelling approach and are guaranteed to find an optimal solution to a C&P problem instance. Although these methods produce optimal packing layouts, they are deemed impractical with respect to solving realistically sized problem instances.

One of the earliest exact C&P solution approaches in the literature is the *column generation method* proposed by Gilmore and Gomory in a series of papers published during the 1960s [64, 65, 66]. They formulated the problem (the one-dimensional bin packing problem [64, 65] or the two-dimensional bin packing problem [66]) as a (linear) integer programming problem which consists of enumerating all the possible combinations of cuts or patterns formed by the items within the bin and determining the number of times each pattern is to be produced in order to satisfy a given demand. Gilmore and Gomory realised the inherent difficulty of solving this integer programming problem directly, in terms of computational efficiency, due to the large number of patterns that is involved in the formulation. They thus reduced the computational difficulty by limiting the number of enumerations necessary through a column generation strategy. The procedure starts by considering a manageable part of the problem (specifically, involving a small number of patterns), solving that part, then adding new feasible patterns to that part, if necessary. The enlarged problem is further solved and the process is repeated until a satisfactory solution to the entire problem is found.

In the two-dimensional case, Gilmore and Gomory restricted the problem by introducing the guillotine-cut constraint. This reduces the number of permissible patterns and also yields a more tractable problem. The column generation technique was also utilised by Scheithauer [142] in an attempt to solve the container and multi-container loading problems to optimality. Although this method is a valid approach, it was found impractical in terms of solving realistically sized C&P problem instances.

In 1977, Christofides and Whitlock [40] developed a *tree-search algorithm* for solving the constrained two-dimensional cutting problem[2] to optimality. The cutting process in the tree-search algorithm is represented by a data structure called a *tree*, in which each node represents a state of the rectangular stock material after cutting has taken place and a branching from one node to another represents a cut. The algorithm applies a transportation routine[3] to optimally allocate pieces to the rectangle at any node. It also limits the size of the tree search by imposing some conditions on the cutting. In 1997, Hifi and Zissimopoulos [84] proposed an improvement to the exact algorithm of Christofides and Whitlock. Their improved algorithm is based on a new branching strategy and a new way of solving the transportation problem at each internal node of the tree. In their experimental study, Hifi and Zissimopoulos showed that these modifications contribute significantly to the effectiveness of the algorithm in respect of time complexity.

In 1998, Martello and Vigo [123] designed a *branch-and-bound algorithm* for finding an optimal solution to the two-dimensional bin packing problem. The algorithm adopts a nested branching scheme. A main branching tree assigns items to bins without specifying their positions in these bins, while a heuristic or an inner branch-decision tree tests, at certain decision nodes, the feasibility of packing the items into a bin, and determines the placing of the items when the answer is positive. The items to be packed are initially sorted in order of non-increasing area, and a first incumbent solution is heuristically obtained. At each decision node, the next free item is assigned to all the initialised bins in turn. When an item i is assigned to a bin, the feasibility of packing item i and all items already assigned to the bin is heuristically checked. Infeasible nodes are immediately discarded, while a heuristic process is performed to determine the placement of the items within the bin in the case of feasible nodes. The inner branch-decision tree is only executed in the case where the heuristic process fails to provide a feasible packing. The inner branching generates all possible ways of packing the items into the bin according to the *left-most downward* principle — an item is placed left and down as far as possible within the bin. Any node representing a candidate solution that is better than the incumbent is stored as the best solution found so far, and the search continues until an optimal solution is obtained.

Authors who also proposed exact approaches for the bin packing problem include Pisinger and Sigurd, who developed a *branch-and-price* approach for the two-dimensional single-size bin packing problem [136] as well as for the variable-size bin packing problem [135], and Belov and Scheithauer [17], who designed a *branch-and-cut-and-price algorithm* for the one- and two-dimensional bin packing problems. The branch-and-price approach is a branch-and-bound method combined with a column generation technique, while the branch-and-cut-and-price algorithm is a combination of a branch-and-price approach and a cutting plane strategy.

In 2003, Martello *et al.* [122] developed an exact algorithm for the strip packing problem, which is based on the branch-and-bound scheme presented by Martello and Vigo [123]. They proposed a relaxation problem for the two-dimensional strip packing problem, to which they referred as the *one-dimensional contiguous bin packing problem*. This problem is solved according to the

[2]In the constrained two-dimensional cutting problem, each piece is associated with a value and a bound delimiting the maximum number of times it should be cut from the rectangular stock material.

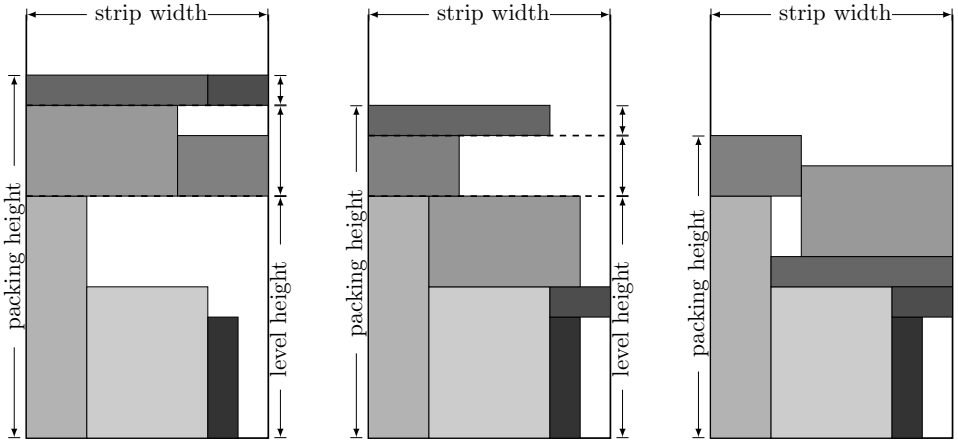
[3]The transportation problem, in this case, is concerned with finding a cutting pattern whose value is maximum, while the respective constraints are satisfied [40, 84].

branch-and-bound algorithm described above. The use of effective bounds made it possible for them to solve test instances from the literature involving up to 200 items. Cui *et al.* [47] and Kenmochi *et al.* [107] also proposed exact algorithms for the two-dimensional strip packing problem, all based on a branch-and-bound strategy. Cui *et al.* considered the case where the guillotine-cut constraint is required and rotations are allowed, while Kenmochi *et al.* solved instances of the strip packing problem that allowed for either oriented or rotational packing.

2.2.2 Heuristic C&P Solution Approaches

Since C&P problems are NP-hard[4], exact methods are often unable to solve realistically sized problem instances within acceptable time frames. Hence there has been considerable interest in techniques that provide satisfactory, yet not necessarily optimal, solutions rapidly. One class of such techniques is the class of *heuristics*, which contains techniques that produce solutions in a reasonable time frame that are considered good enough for the problem instance at hand, but these techniques usually do not return optimal solutions.

In the context of packing problems, heuristics are algorithms that arrange a given sequence of items directly within a strip or bin by following a fixed set of rules. While the packing layouts produced by these methods are feasible, they are not necessarily optimal. Several strip packing heuristic algorithms have been suggested in the literature. They may be classified into the subclasses of *plane algorithms*, *pseudolevel algorithms* and *level algorithms*. Members of the class of plane algorithms are able to pack items anywhere in the space defined by the boundaries of the strip. In pseudolevel algorithms, on the other hand, the packing of items is restricted by horizontal levels in which no packed items intersect any level boundary. A level is delimited by two parallel, horizontal lines joining the two unbounded vertical sides of the strip. Level algorithms may, in fact, be considered a subclass of pseudolevel algorithms where at least one edge of each item packed must coincide with the lower boundary of a level. An example of a solution obtained by means of each of these types of heuristics, when applied to the same packing instance, is shown in Figure 2.2.



(a) *Level solution* (b) *Pseudolevel solution* (c) *Plane solution*

FIGURE 2.2: *Examples of strip packing solutions resulting from the application of level, pseudolevel and plane algorithms. Dashed lines define the level boundaries.*

[4]A problem is NP-hard if solving it in polynomial time would make it possible to solve all problems in the class of NP (non-deterministic polynomial time) problems in polynomial time. Since the one-dimensional bin packing problem has been shown to be NP-hard [119, 122], all related cutting and packing problems are also NP-hard.

The majority of level algorithms proposed in the literature originated from algorithms designed for the one-dimensional bin packing problem by Johnson *et al.* [100, 101]. The items in the list are sorted according to non-increasing height and are packed in this order into a strip or bin by forming a sequence of levels. The first level is the bottom of the bin and each subsequent level is determined by a line coinciding with the top-most boundary of the highest item placed on the previous level, extending from one side of the strip to its opposite side.

The *next-fit decreasing height* (NFDH) algorithm and *first-fit decreasing height* (FFDH) algorithm are among the earliest level algorithms for the two-dimensional strip packing problem, first proposed by Coffman *et al.* [42] in 1980. The NFDH algorithm starts by packing the first item in the bottom left-hand corner of the strip and then fills the incumbent level from left to right with subsequent items until there is insufficient space to accommodate a particular item. At that point, packing proceeds on a new level and no further packing is allowed on the current level. In the FFDH algorithm, however, items are packed on the lowest level in which they fit. If none of the existing levels can accommodate the next item, a new level is initialised. The *best-fit decreasing height* (BFDH) algorithm, proposed by Coffman and Shor [44] in 1990, and the *worst-fit decreasing height* (WFDH) algorithm, proposed by Ortmann [131] in 2010, are variants of the FFDH algorithm. As in the FFDH algorithm, both algorithms allow any existing levels to be revisited, except that the BFDH algorithm packs items on the level in which they fit and which achieves the minimum residual horizontal space (the space remaining between the item's right-hand edge and the right-hand boundary of the strip), while the WFDH algorithm selects the level with the maximum residual horizontal space instead.

Authors who proposed level algorithms for the strip packing problem include Martello *et al.* [122], who designed a heuristic strategy (called JOIN) for finding a good initial solution for their exact approaches, and Lodi *et al.* [120], who introduced a new level algorithm based on the celebrated knapsack problem for solving the two-dimensional strip packing problem in 1999. A recent level algorithm for the two-dimensional strip packing problem is the *best two-fit decreasing height* (B2FDH) algorithm proposed by Ortmann [131] in 2010. It is similar to the FFDH algorithm, except that before packing on the next level, an attempt is made to replace the last item on the current level with two unpacked items that have a sum of widths or combined area greater than its width or area.

Pseudolevel algorithms pack items on levels, as do all level algorithms, but the difference is that these algorithms allow items to be packed anywhere in the area delimited by the boundaries of levels. That is, items may be stacked on top of any floor-packed items or they may be placed on the ceiling of a level as long as the overlapping constraint is not violated.

During the late 1990s, Lodi *et al.* [120, 121] developed the *floor-ceiling* (FC) algorithm for solving the single bin size bin packing problem. It is essentially a two-phase approach in which the first phase consists of solving a two-dimensional strip packing problem. The FC algorithm may pack items from left to right on the floor of a level (the horizontal line drawn through the bottom edge of the left-most item) or from right to left on the ceiling of a level (the horizontal line drawn through the top edge of the left-most item). All floor or ceiling packing is realised according to a best-fit strategy — items are assigned to the floor or ceiling of a level with minimum residual horizontal space.

Ntene [128] proposed the *size-alternating stack* (SAS) algorithm in 2007. The list of items is partitioned into two sublists, the first sublist contains narrow items (items of height greater than their width) while the second sublist is populated by wide items. The filling process of a level is achieved by alternating between narrow and wide items — if a narrow item initialises a level, then the subsequent item packed to its right is a wide item, and *vice-versa*. Items from the same list are allowed to be stacked on top each other. In 2010, Ortmann [131] suggested

an improvement to the SAS algorithm. In their resulting SASm algorithm, narrow items are allowed to be stacked on top of wide items and are also allowed to be placed next to each other.

Ortmann [131] also proposed three pseudolevel algorithms, called the *stack level* (SL), the *stack ceiling* (SC), and the *stack ceiling with re-sorting* (SCR) algorithms, which are all based on the stacking strategy of the SAS algorithm. In the SL algorithm, the first item in the list initialises the first level and subsequent items are packed in a best-fit manner. Once this process has been completed, the area of free space above the items of proportional height is identified and filled in a first-fit manner. In the SC and SCR algorithms, a level is filled in a first-fit manner. When no further items fit on the floor of a level, the stacking process begins (the SCR algorithm first re-sorts the items by non-increasing width and height). Stacking occurs downwards from ceiling-packed items. The tallest (widest for the SCR algorithm) unpacked item is assigned to the ceiling, then a search is carried out to identify items that may be stacked below it. This operation is performed for all ceiling-packed items.

In contrast to the two previous classes of heuristics, plane algorithms pack items anywhere in the strip without level restriction. The packing solutions returned by plane algorithms are therefore not guaranteed to be guillotineable.

Sleator [144] appears to be the first author who proposed a plane heuristic for the two-dimensional strip packing problem in 1980. Sleator's algorithm starts by packing all items of width larger than half the strip width on top of each other. The remaining items are ordered by non-increasing height. The first unpacked item is placed on top of the last packed item (left-justified), thus forming a level which is filled in a next-fit manner. Once this filling process has been completed, the strip is partitioned into two equal halves. The top edge of the tallest item in either half-strip defines the left and right baselines. The algorithm selects the half-strip with lower baseline and fills it in a next-fit manner. This process continues until no unpacked items remain.

Coffman *et al.* [42] proposed the *split-fit* algorithm in 1980. They defined a parameter m as the largest integer for which all items have width less than or equal to $1/m$ (the dimension of items and the strip width are normalised to 1). The list of items to be packed is partitioned into two sublists, consisting of items of width greater than $1/(m+1)$ and less than or equal to $1/(m+1)$, respectively. An attempt is made to pack items in the first list in a first-fit manner. The resulting levels are rearranged such that all levels with width greater than $(m+1)/(m+2)$ are placed below all levels with width less than or equal to $(m+1)/(m+2)$. An unoccupied area of width $1/(m+2)$ thus emerges adjacent to the latter set of levels. This area is filled with the items from the second list in a first-fit manner without overlapping its boundaries. The remaining items are packed above the top-most level according to the FFDH algorithm.

Other plane algorithms proposed for the strip packing problem include the class of *bottom-up left-justified* (BL) algorithms introduced by Baker *et al.* [11] in 1980, the *split* and *mixed* algorithms of Golan [69] in 1981, the *up-down* algorithm of Baker *et al.* [10], also dating from 1981, and the *best-fit* algorithm proposed by Burke *et al.* [29] in 2004.

2.2.3 Metaheuristic C&P Solution Approaches

A second type of practical approximate algorithm for solving strip packing problems is the class of *metaheuristics*. Algorithms in this class operate at a higher level of flexibility than heuristics and provide sufficiently near-optimal solutions within reasonable time frames. Metaheuristic techniques employ a trade-off between randomisation and local search in order to avoid entrapment at local optima. Metaheuristic algorithms may be classified into *trajectory-based methods*

(or *single-point techniques*) and *population-based algorithms*. Trajectory-based methods deal with a single candidate solution at a time and perform search processes which describe a trajectory in the search space over time. This class of algorithms includes the celebrated methods of *tabu search* and *simulated annealing*. The working of population-based metaheuristics, on the other hand, is based on a set of candidate solutions at any one time, called a *population* of solutions. They perform the search process by describing the simultaneous evolution of these solutions in the search space over time. Well-known examples of such algorithms are *genetic algorithms* and the *method of ant colony optimisation*. Each of these metaheuristics follows its own search philosophy. Some metaheuristics are inspired by natural processes, such as biological evolution or foraging behavior of animals, while others are extensions of simple heuristics, such as local search and constructive heuristics.

In the context of packing problems, the search space consists of all possible feasible arrangements of items in the strip or bin, and metaheuristics aim to explore this space by applying a set of adaptive rules in order to identify the best packing order of items and to generate a near-optimal packing layout corresponding to this packing order.

The most widely studied algorithms in the class of population-based metaheuristics for solving the strip packing problem are *genetic algorithms* (GAs). Algorithms in this class, first proposed by Holland [87] in 1975, are based on the principle of biological evolution. Solutions are encoded in an appropriate format (often in the form of binary strings) and a set of operators, including selection, recombination and mutation, is iteratively applied to a population of candidate solutions in an attempt to reach a near-optimum. Typically, a GA starts with an initial, randomly generated set of candidate solutions that are evaluated according to a so-called fitness function. The fittest solutions are then selected from the current population and are allowed to reproduce, by applying recombination and mutation operators to them, in order to form a new generation of candidate solutions. The subsequent generation is similarly altered during the next iteration and the process is repeated until a stopping criterion is satisfied.

Hopper and Turton [89, 90, 91] conducted an extensive review of the application of metaheuristics for the strip packing problem and distinguished between three types of solution approaches involving GAs in the literature. Approaches in the first category concentrate on *hybrid techniques*, whereby a GA is combined with some placement routine. These methods make use of a particular type of coding of solutions — the standard one is known as a *permutation*, which specifies the order of packing the items one by one. An example of a permutation is a list of items sorted according to non-increasing height. At each iteration, a GA is applied to carry out a search in the space of the encoded solutions in order to determine suitable permutations according to which the items should be packed, and then a placement or decoding routine is called upon to transform this sequence into a complete packing layout. Examples of solution approaches in this class include the hybrid approaches of Jakobs [99] and of Liu and Teng [118], in which heuristics in the class of bottom-left algorithms are used as decoding routines. Other examples are contained in the work of Hopper and Turton [91] in which they evaluated the performance of two hybrid GAs, the first one in conjunction with the BL-algorithm of Jakobs [99] and the second one in conjuction with the BLF-algorithm of Chazelle [37], as well as the combination of a recursive heuristic and a GA proposed by Zhang *et al.* [164].

The second type of solution approach involving GAs incorporates some of the layout information in the encoded solutions and employs an additional rule to generate the complete layout. The solution encoding scheme used in the hybrid algorithms described above reveals only the order in which the items are to be packed, with no further information on the position or geometric placement of the items within the layout. This latter detail is hidden in the decoding routine. The encoded solutions used in the approaches of the second category, however, also contain

some information about the position of the items in the strip. Examples of algorithms in this category include the GAs proposed by Kröger [113] and by Hwang *et al.* [93], which are all based on the notion of a *slicing tree structure*. The leaf nodes of this structure correspond to the items to be packed, while all interior nodes define successive cuts. Figure 2.3 contains an example of a slicing tree structure where v (h, respectively) represents a vertical (horizontal, respectively) cut and the numbers 1 to 6 represent the items.

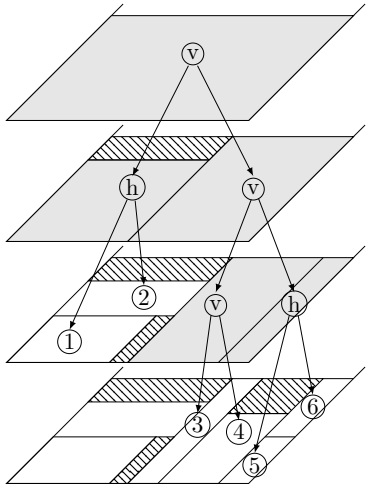


FIGURE 2.3: *An example of a slicing tree structure.*

A third category of GA solution approaches for the strip packing problem attempts to solve the problem directly in the space of the fully defined layout, without any encoding of solutions. A relatively small body of literature is available for this type of GA, with Ratanapan and Dagli [138] developing the object-based evolutionary algorithm for the two-dimensional bin packing problem in 1997, Bortfeldt and Gehring [26] suggesting a hybrid GA for the container loading problem in 2001, and Bortfeldt [25] proposing an adaptation of the latter algorithm for solving the two-dimensional strip packing problem in 2006.

The most common algorithm in the class of trajectory-based metaheuristics that has been applied to C&P problems is *simulated annealing* (SA), first proposed by Kirkpatrick *et al.* [109]. It mimics the process of physical annealing of solids, in which a solid is heated and then allowed to cool slowly until it achieves a regular crystalline structure. SA interprets slow cooling as a slow decrease in the probability of accepting worse solutions over time as it explores the solution space. This probability is a function of a parameter, called the *temperature*. Typically, the SA algorithm starts with a random initial solution and a relatively large initial value of the temperature parameter. At each iteration, the algorithm randomly selects a neighbourhood solution of the current one and probabilistically decides between accepting it as new current solution or rejecting it. A better solution is always accepted, while a worse solution is accepted according to the defined probability. The temperature is decreased during the search process[5] and the process continues until a termination condition is met.

The early literature on the application of SA to C&P problems mainly focused on the application of SA to the pallet loading problem or to the cutting stock problem. Dowsland [52] experimented with SA in the context of pallet loading problems involving identical and non-indentical items in 1993, while Lai and Chan [114] developed a hybrid search approach in 1997 whereby SA

[5]The cooling schedule or the way in which the temperature decreases is usually critical in respect of the success of the technique. There is, however, no specific rule that defines the best cooling schedule and initial temperature value — these are typically determined empirically in the context of the particular problem that has to be solved.

is combined with a placement strategy in order to minimise the trim loss of the cutting stock problem. Faina [62] proposed a SA approach for solving two-dimensional cutting problems involving guillotine as well as non-guillotine constraints.

Application of SA to the two-dimensional strip packing problem was, to the best knowledge of the author, first considered by Hopper and Turton [91] in 2001. In addition to the two hybrid GAs mentioned above, they also developed and evaluated the performance of two hybrid SA algorithms, the first being a hybrid between SA and the BL-algorithm of Jakobs [99] and the second approach being a hybrid between SA and the BLF-algorithm of Chazelle [37]. The SA algorithm adopted in their approaches may be interpreted as a special case of their GAs in which only a single-individual population, represented by a permutation, is considered and where one offspring solution is created (by means of a mutation) at each iteration. The selection process is omitted and the replacement operator is based on a given probability. The mutation operator in this case consists of either swapping two randomly selected items or complementing the rotation variable of one randomly selected item. The BL-algorithm and the BLF-algorithm transform the permutation into a complete layout. Hopper and Turton showed that the SA combined with the BLF-algorithm achieved the best layout quality over all problem instances considered by them.

In 2009, Burke *et al.* [31] presented a SA enhancement of the best-fit heuristic (the one that they proposed in their earlier work [29]) for the strip packing problem. The proposed hybrid strategy consists of two phases. During the first phase, the best-fit heuristic is invoked to pack $n-m$ items, where n denotes the total number of items to be packed and m ($0 < m < 50$) is the number of items that are to be packed during the second phase. The second phase involves the packing of the remaining items by applying the SA hybridised with the BLF heuristic of Hopper and Turton [91]. This strategy was motivated by the observation that the best-fit heuristic produces high-quality solutions in the range of medium to large benchmark problem instances, but performs poorly on smaller-sized problem instances. On the other hand, the SA bottom-left method yields good results for smaller-sized problem instances. Burke *et al.* thus combined the strengths of both approaches in order to achieve good solution quality over the entire range of existing benchmark instances available in the literature.

The second common trajectory-based metaheuristic in the packing problem literature is *tabu search* (TS), first proposed by Glover [67]. It is an iterative procedure that progressively improves an initial solution toward successively better solutions by applying a controlled series of movements *via* a dynamic list of non-forbidden moves (*i.e.* avoiding moves in a so-called *tabu list*). Typically, the TS algorithm starts with a randomly generated initial solution and an empty tabu list. At each iteration, the best neighbourhood solution of the current solution that does not belong to the tabu list is selected as new current solution. Reversal of this move is inserted into the tabu list and the process continues until a termination condition is met. There are no specific rules that determine the best value of the neighbourhood size or the length of the tabu list. These values are typically determined empirically. Authors who have applied the tabu search technique in the context of the two-dimensional rectangular packing problem include Lodi *et al.* [120], who proposed a unified tabu search framework for the two-dimensional bin packing problem, Alvarez-Valdés *et al.* [3, 6], who developed a tabu search algorithm for guillotine and non-guillotine cutting problems, Pureza and Morabito [137], who conducted experiments with the TS algorithm for the pallet loading problem, and Burke *et al.* [31], who presented a TS enhancement of the best-fit heuristic (the one that they proposed in their earlier work [29]), in addition to the hybrid SA described earlier, for the two-dimensional strip packing problem.

Other currently popular state-of-the-art algorithms for the strip packing problem include the reactive *greedy randomised adaptive search procedure* (GRASP) of Alvarez-Valdés *et al.* [5] (2006),

the *intelligent search algorithm* (ISA) of Leung *et al.* [116] (2011), the *simple randomised algorithm* (SRA) of Yang *et al.* [162] (2013) and the *improved algorithm* (IA) of Wei *et al.* [158] (2016).

2.3 Dissertation Scope

In this dissertation, the focus is on the two-dimensional strip packing problem, in which all items are rectangular and the objective is to minimise the strip packing height. The feasibility of a solution in terms of a guillotine-cut is optional: this constraint may or may not be required, depending on the specificity of the existing algorithms. Items have a fixed orientation (that is, rotation is disallowed). According to Ntene's subtypology [128], the C&P problems considered in the remainder of this dissertation may therefore be classified as:

2D	R	SP	Off	MiS	0,0,0,*

.

This representation indicates that two-dimensional rectangular items are to be packed into a strip, that the entire list of items is known before the packing process commences, that the required strip packing height should be minimised, that rotation is not allowed, that there is no restriction on the placement of items, that the shapes of the items may not be modified, and that the guillotine-cut constraint may or may not be required.

In terms of solution methodologies, the focus is on the application of heuristic and metaheuristic algorithms for solving the strip packing problem. Five algorithms belonging to the class of heuristic techniques are considered. These heuristics are employed as decoding algorithms in hybrid metaheuristics. Furthermore, solution approaches involving GAs in the first and third categories, as well as hybrids between SA and the current state-of-the-art strip packing metaheuristic techniques, are studied.

2.4 Chapter Summary

The aim of this chapter has been to review the literature on C&P problems, and to delimit the scope of this dissertation. The typologies of Dyckhoff [55], Wäscher *et al.* [157], Lodi *et al.* [120], and Ntene [128] for such problems were reviewed in §2.1.1. This was followed by a summary of basic and popular types of C&P problems encountered in various application areas in §2.1.2. After the discussion on problem typologies, the different methods available for solving C&P problems, namely exact methods, heuristic techniques and metaheuristic techniques, were reviewed in §2.2. This made it possible to clarify the scope of C&P problems and their associated solution methodologies considered in the remainder of this dissertation (§2.3).

CHAPTER 3

Strip Packing Heuristics

Contents

As mentioned in §1.3, one of the aims of this study is to characterise different SPP heuristic and metaheuristic algorithms documented in the literature in terms of their strengths and weaknesses in respect of a large set of strip packing benchmark instances. In this chapter, five algorithms belonging to the class of heuristic techniques which are employed in a comparative study later in this dissertation, are discussed in detail. A description of each algorithm is provided, and this is followed by a pseudocode listing of the procedure together with a worked example of its application.

The SPP instance C1-P3 of Hopper and Turton [90], called the set $\mathcal{I}$ in the remainder of this dissertation, is used to illustrate the working of the algorithms presented in this chapter and the

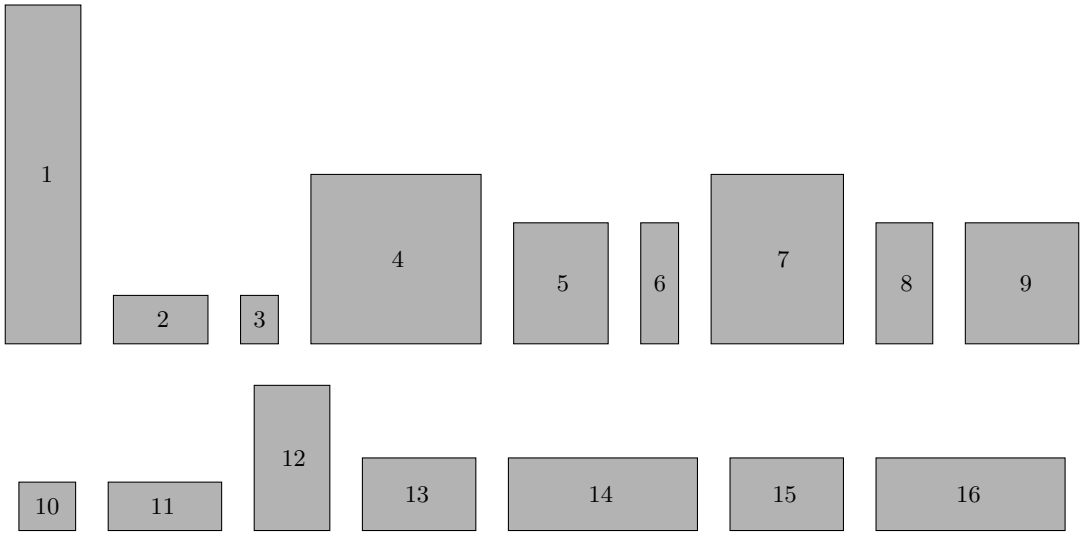


FIGURE 3.1: *The items in $\mathcal{I}$ used for illustrative purposes.*

next. The items in $\mathcal{I}$ are shown in Figure 3.1 and their dimensions are listed in Table 3.1. The width of the strip is specified as 20.

Item, $\mathcal{I}_i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Height, $h(\mathcal{I}_i)$	14	2	2	7	5	5	7	5	5	2	2	6	3	3	3	3
Width, $w(\mathcal{I}_i)$	4	5	2	7	5	2	9	3	6	3	6	4	6	10	6	10

TABLE 3.1: *Dimensions of the rectangular items in the set $\mathcal{I}$.*

3.1 The Modified Best-Fit Decreasing Height Algorithm

One of the earliest heuristic strip packing algorithms is the *best-fit decreasing height* (BFDH) algorithm proposed by Coffman and Shor [44] in 1990. The BFDH algorithm packs a list of items, initially sorted by non-increasing height, into consecutive levels by filling a level from left to right. It packs each item into the level in which it fits and achieves the minimum residual horizontal space — the space remaining between the item's right-hand edge and the right-hand boundary of the strip. A new level is only initialised if no existing levels can accommodate the item.

In 2006, Bortfeldt [25] modified the BFDH algorithm, calling the modification the BFDH* algorithm, in such a way that items are packed on levels by utilising the remaining space between the top edge of packed items and the ceilings of their levels. The algorithm starts by filling a level from left to right according to the BFDH packing strategy. Each time an item is packed into a level, the remaining space on the floor is evaluated. If that space is too small to accommodate the thinnest unpacked item, areas of free space are defined above the items packed on that level (as illustrated in Figure 3.2). Each free space is then filled, from left to right, with unpacked items of the largest area that can fit into it. Once this free space filling process has been completed, the algorithm returns to the BFDH level packing strategy. The entire process is repeated until all items are packed. A pseudocode representation of the BFDH* algorithm is shown in Algorithm 3.1.

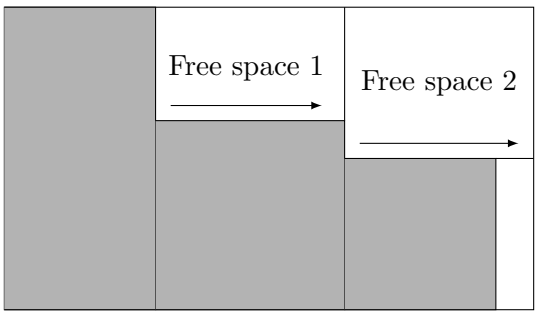


FIGURE 3.2: *The free space utilisation by the BFDH* algorithm.*

Example 1 *By sorting the instance in Table 3.1 according to decreasing height, the ordered list $\mathcal{I}' = \{\mathcal{I}_1, \mathcal{I}_4, \mathcal{I}_7, \mathcal{I}_{12}, \mathcal{I}_5, \mathcal{I}_6, \mathcal{I}_8, \mathcal{I}_9, \mathcal{I}_{13}, \mathcal{I}_{14}, \mathcal{I}_{15}, \mathcal{I}_{16}, \mathcal{I}_2, \mathcal{I}_3, \mathcal{I}_{10}, \mathcal{I}_{11}\}$ results. Item $\mathcal{I}_1$ initialises the first level, and items $\mathcal{I}_4$ and $\mathcal{I}_7$, the subsequent items in the list, are packed adjacent to it. Since no remaining space is available on the floor of the first level, two areas of free space are defined. The first space is between the top edge of $\mathcal{I}_4$ and the ceiling of the first level, while the second area of free space is between the top edge of $\mathcal{I}_7$ and the boundary of the incumbent level. The unpacked item with the largest area is $\mathcal{I}_9$ and is stacked onto $\mathcal{I}_4$. Item $\mathcal{I}_8$*

does fit into the remaining region of the first free space and is therefore packed adjacent to $\mathcal{I}_9$*. Items* $\mathcal{I}_{14}$ *and* $\mathcal{I}_{16}$ *do not, however, fit into the second free space. This allows* $\mathcal{I}_5$ *to be stacked onto* $\mathcal{I}_7$*. Item* $\mathcal{I}_6$ *is the item with the largest area among the unpacked items that fits into the remaining region of the second free space. Therefore, it is packed adjacent to* $\mathcal{I}_5$*.*

There is no further space in the first level for $\mathcal{I}_{12}$*; hence it initialises a second level. This level has sufficient space for items* $\mathcal{I}_{13}$ *and* $\mathcal{I}_{14}$*. Therefore, these items are packed adjacent to* $\mathcal{I}_{12}$ *resulting in a zero-waste space between the right-hand edge of* $\mathcal{I}_{14}$ *and the right-hand boundary of the strip. The areas above* $\mathcal{I}_{13}$ *and* $\mathcal{I}_{14}$ *are designated free space for further packing. Item* $\mathcal{I}_{15}$ *is the first unpacked item with the largest area that fits into the first region of free space and is stacked onto* $\mathcal{I}_{13}$*. Item* $\mathcal{I}_{16}$*, the next unpacked item in the list, fits into the second free space and is therefore packed above* $\mathcal{I}_{14}$*. The last four remaining items,* $\mathcal{I}_2$*,* $\mathcal{I}_3$*,* $\mathcal{I}_{10}$ *and* $\mathcal{I}_{11}$*, do not fit into any of the existing levels and are hence packed into a new level, resulting in the solution shown in Figure 3.3.* ■

Algorithm 3.1: **Bortfeldt's modified best-fit with decreasing height algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.
Output: A feasible packing of the items in $\mathcal{P}$ into a strip of width W.

```
1  sort the list of items P by decreasing height;
2  make a copy A of P, sort it by decreasing area;
3  NumLevels ← 1, the number of existing levels;
4  while there are unpacked items do
5      set P_F as the first unpacked items in P;
6      find MinResLevel, the level with minimum residual horizontal space;
7      if P_F does not fit onto any existing levels then
8          NumLevels ← NumLevels + 1, pack P_F into level NumLevels;
9          h(NumLevels) ← h(P_F) , ω(NumLevels) ← W − ω(P_F);
10         remove the equivalent item from A;
11     else
12         pack P_F into level MinResLevel;
13         ω(MinResLevel) ← ω(MinResLevel) − ω(P_F);
14         determine the thinnest unpacked item P_N in P;
15         if ω(MinResLevel) < ω(P_N) then
16             define the regions of free space above the items in MinResLevel;
17             while empty regions of free space and unpacked items remain do
18                 select the left-most empty region of free space;
19                 pack items in A onto the floor of the region until no more fit;
20                 remove the corresponding items from P;
```

3.2 The Bottom-Left Algorithm

The class of *bottom-up left-justified* (BL) algorithms was first introduced by Baker *et al.* [11] in 1980. Algorithms in this class pack items in the lowest possible space and left-justified. Such algorithms preserve the so-called *BL-condition* — no packed items can be shifted further to the bottom or to the left. Different members of class of BL algorithms exist in the literature.

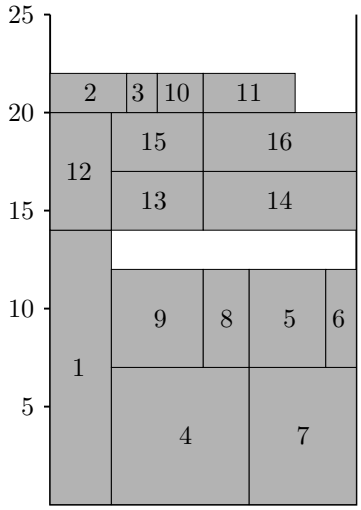


FIGURE 3.3: *Result obtained when packing items in $\mathcal{I}$ using the BFDH* algorithm. The resulting packing height is* $\mathtt{H} = 22$.

Chazelle [37], Hopper and Turton [91], Jakobs [99], and Liu and Teng [118] all investigated members of this class. In this dissertation, the BL heuristic suggested by Liu and Teng [118] is adopted throughout. This algorithm has been shown to be more effective than that of Jakobs, and its execution time is considerably smaller than that of Chazelle ([29, 91, 118]).

The algorithm starts by sorting the list of items in a predetermined manner and proceeds to pack the items one by one. The process of packing an item begins by placing it in the top-right corner of the strip and then making successive moves by repeatedly sliding it as far as possible to the bottom and then as far as possible to the left (see Figure 3.4). During the sliding process, the algorithm gives priority to the vertical movements.

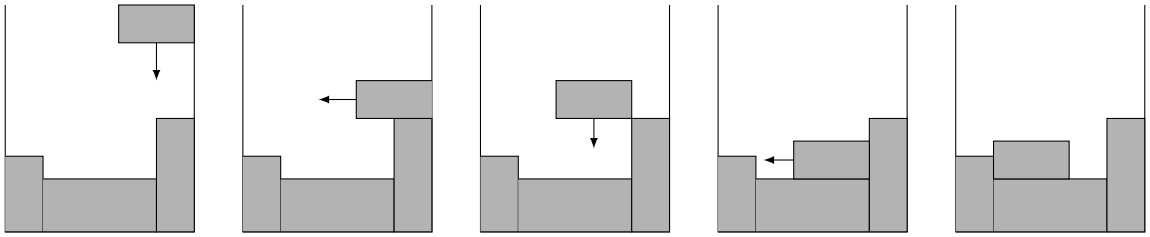
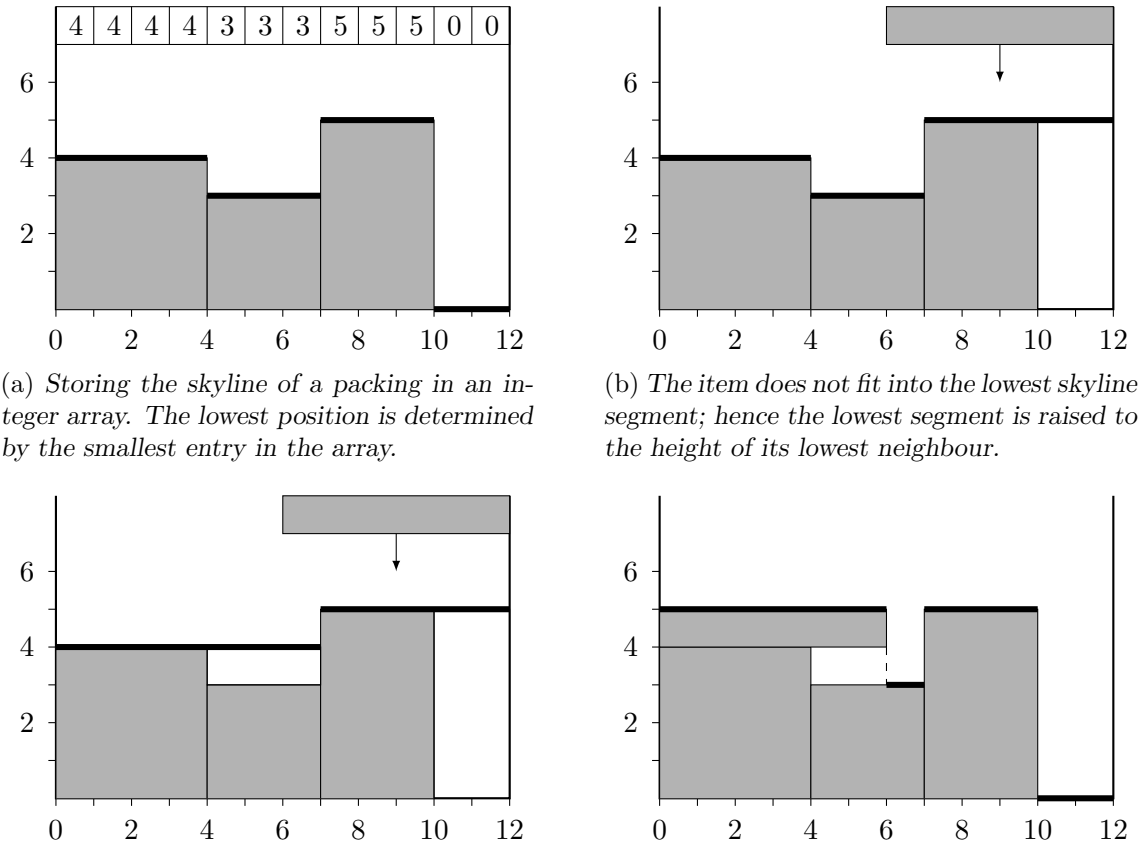


FIGURE 3.4: *The improved bottom-left method of Liu and Teng [118].*

The concept of a *skyline*, as proposed by Burke *et al.* [29], makes it possible to represent the shape of the packing during the execution of the BL algorithm. An array is employed to record the height of packed items at each unit interval after each packing. The coordinate of the lowest and left-most position is determined by the location of the smallest-valued entry of the array, while its width is equal to the length of the consecutive array entries of the same value. In Figure 3.5(a), for example, the lowest available space is located at $x = 10$ and has a width of two. The union of all location levels defines the skyline (represented by the thick lines in Figure 3.5(a)). The BL algorithm starts by finding the lowest position of the strip that can accommodate an item. In the case where the lowest position is narrower than the item is wide, it is raised to the height of its shortest neighbour. This process is repeated until the lowest space is wide enough to accommodate the item. The item is then placed left-justified in this lowest position. This procedure is illustrated in Figure 3.5(b)–(d). After an item has been packed, the

skyline is updated and the process continues until no unpacked items remain. A pseudocode listing of the algorithm may be found in Algorithm 3.2.



(a) *Storing the skyline of a packing in an integer array. The lowest position is determined by the smallest entry in the array.*

(b) *The item does not fit into the lowest skyline segment; hence the lowest segment is raised to the height of its lowest neighbour.*

(c) *The lowest temporary skyline segment remains too narrow for the item, resulting in it being raised to the height of its lowest neighbour.*

(d) *The lowest temporary skyline segment is wide enough to accommodate the item and the skyline is updated to reflect the position of the packed item.*

FIGURE 3.5: *(a) Representation of a packing by means of a skyline. (b)–(d) The process followed to pack an item during execution of the BL algorithm. The lowest available position is sought for packing and a copy of the skyline (represented by the thick lines) is modified until the lowest segment is wide enough for the item to be packed into it.*

Example 2 *By sorting the instance in Table 3.1 according to decreasing height, the ordered list* $\mathcal{I}' = \{\mathcal{I}_1, \mathcal{I}_4, \mathcal{I}_7, \mathcal{I}_{12}, \mathcal{I}_5, \mathcal{I}_6, \mathcal{I}_8, \mathcal{I}_9, \mathcal{I}_{13}, \mathcal{I}_{14}, \mathcal{I}_{15}, \mathcal{I}_{16}, \mathcal{I}_2, \mathcal{I}_3, \mathcal{I}_{10}, \mathcal{I}_{11}\}$ *results. The first item,* $\mathcal{I}_1$*, is placed in the bottom-left corner of the strip and the skyline is updated to consist of two segments; the part above* $\mathcal{I}_1$ *and the part to its right. The latter segment is the lowest, and is wide enough to accommodate items* $\mathcal{I}_4$ *and* $\mathcal{I}_7$*. These items are thus packed in that space next to* $\mathcal{I}_1$ *as shown in Figure 3.6. The second skyline segment now consists of the part that spans the top edges of* $\mathcal{I}_4$ *and* $\mathcal{I}_7$*, and remains the lowest position for packing. The sum of the width of the next four items in the list,* $\mathcal{I}_{12}$*,* $\mathcal{I}_5$*,* $\mathcal{I}_6$ *and* $\mathcal{I}_8$*, is smaller than the width of the lowest position; resulting in their packing in that space next to each other and left-justified.*

The packing now consists of four skyline sections. The section between the right-hand edge of $\mathcal{I}_8$ *and the right-hand boundary of the strip is the lowest, but too narrow to accommodate* $\mathcal{I}_9$*. The section is therefore raised to the height of the middle section. This combined section is now*

Algorithm 3.2: **Bottom-left algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.
Output: A feasible packing of the items in $\mathcal{P}$ into a strip of width W.

```
1  sort the list of items in some manner, if required;
2  initialise the skyline S to reflect the space available for packing;
3  for i ← 1 to n do
4      make a copy K of the skyline S, Found ← False;
5      while not Found do
6          let s be the index of the skyline segment with the lowest height;
7          if ω(K_s) ≥ ω(P_i) then
8              Height ← h(K_s), Found ← True;
9          else
10             let ℓ the index of the left-hand skyline segment;
11             let r the index of the right-hand skyline segment;
12             if h(K_ℓ) ≤ h(K_r) then
13                 ω(K_ℓ) ← ω(K_ℓ) + ω(K_s), remove index s from the skyline K;
14             else
15                 ω(K_s) ← ω(K_s) + ω(K_r), remove index r from the skyline K;
16     discard the temporary skyline K;
17     move P_i as far to the left as possible at height Height;
18     update the skyline in order to reflect packing of the additional item;
```

the lowest and is wide enough to accommodate item $\mathcal{I}_9$, resulting in its packing there. The next item, $\mathcal{I}_{13}$, is placed adjacent to $\mathcal{I}_9$, filling the remaining space. The skyline is updated to consist of four segments; the parts spanning the top edges of $\mathcal{I}_1$, $\mathcal{I}_{12}$, $\mathcal{I}_9$, and $\mathcal{I}_{13}$, respectively. The second segment of the skyline is the lowest position, but its width is less than that of $\mathcal{I}_{14}$; the segment is therefore raised to the height of the first segment. This new temporary segment is now the lowest, but is still too narrow for $\mathcal{I}_{14}$, resulting in the segment being raised to the height of the top edge of $\mathcal{I}_9$. The incumbent lowest segment, the part above $\mathcal{I}_{13}$, remains too narrow to accommodate $\mathcal{I}_{14}$. The skyline is thus updated to consist of one segment, the part above $\mathcal{I}_9$ with a width of 20, and $\mathcal{I}_{14}$ is packed there, left-justified.

At this stage, the packing consists of three skyline segments. Item $\mathcal{I}_{15}$ fits perfectly on the lowest segment, the part on top of $\mathcal{I}_{13}$, and is stacked onto it. The middle segment is now the lowest position for packing, but is too narrow to accommodate $\mathcal{I}_{16}$. Raising this segment to the height of the segment above $\mathcal{I}_{15}$ yields a segment of the same width as $\mathcal{I}_{16}$. Once $\mathcal{I}_{16}$ has been packed onto this segment, the skyline is reduced to two segments. The segment above $\mathcal{I}_{14}$ is the lowest and is wide enough to accommodate the next three items in the list, $\mathcal{I}_2$, $\mathcal{I}_3$, and $\mathcal{I}_{10}$. The last item, $\mathcal{I}_{11}$, is slided to the lowest and left-most position, resulting in its packing on top of $\mathcal{I}_{16}$. A graphical representation of the final packing may be found in Figure 3.6. ■

A significant weakness of the integer representation of the skyline of a packing is the lack of support for non-integer item dimensions, as there exist SPP instances (such as those considered by Wang and Valenzuela [156]) in which the items have floating-point dimensions. In order to accurately monitor the skyline, an array of quadruples is used to represent each segment of the skyline. This data structure comprises the index value of the skyline segment, its horizontal and

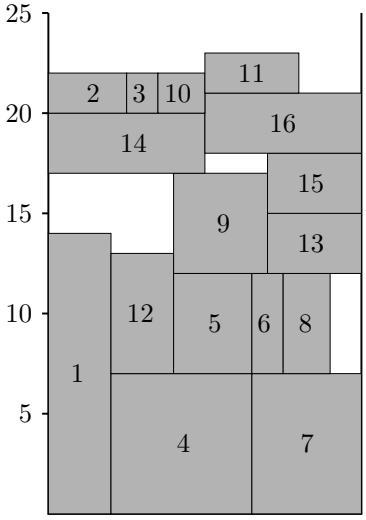
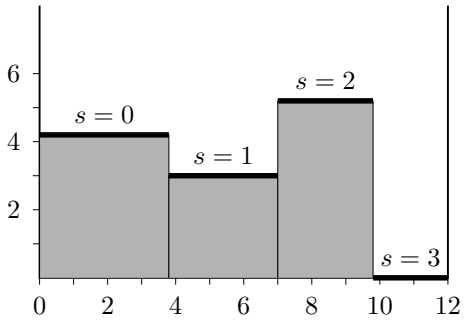


FIGURE 3.6: *Result obtained when packing items in $\mathcal{I}$ using the BL algorithm. The resulting packing height is* $\mathtt{H} = 23$.

vertical coordinates as well as its width (see Figure 3.7). A skyline may, thus, be represented by an array of arrays collectively representing all segments spanning the skyline. Adopting this representation, it is easy to determine which index in the array has the lowest packing location, and it also facilitates the addition of segments to the skyline and the removal of segments from the skyline. Such addition or deletion may be required when a skyline is updated after an item has been packed. If an item $\mathcal{P}_i$ is packed onto a skyline segment s and the item is not as wide as the segment, then an addition must be made to the skyline. The procedure in Figure 3.8 may be followed to add a segment to the skyline and modify the skyline accordingly.



s	X	Y	W
0	0	4.2	3.8

s	X	Y	W
1	3.8	3	3.2

s	X	Y	W
2	7	5.2	2.8

s	X	Y	W
3	9.8	0	2.2

FIGURE 3.7: *A practical approach to storing the skyline of a packing. Each segment s of the skyline is represented by its horizontal coordinate X, vertical coordinate Y, and width W.*

Overhangs might also occur during the packing process (see Figure 3.9). It is, however, important to ensure that the packing of an item yields a left-justified packing, with no further move to the left allowed. Therefore, the possibility of overhangs requires an additional algorithmic step in an attempt to shift the item so as to achieve its left-most position. In order to determine how far to the left an item i may be moved, the locations of all $i-1$ previously packed items have to be tested for possible overlapping. This may be achieved by following the procedure outlined in Figure 3.10. The possibility of this additional move must be considered during each update of the skyline. In the case where an item fits completely under the overhang, no updating is required as it is under an existing skyline segment. If the difference between the item width and the move space is, however, larger than the width of the skyline over which the item is packed, then some segments of the skyline may be deleted and some may be modified.

```
1 S_s ← active skyline segment;
2 s + i ← s + i + 1 for i = 1, ..., |S| (shift all skyline indices from s + 1 to the end, by one
  unit, in order to add the new segment);
3 S_{s+1}.X ← S_s.X + ω(P_i);
4 S_{s+1}.Y ← S_s.Y;
5 S_{s+1}.W ← S_s.W − ω(P_i);
6 S_s.Y ← S_s.Y + h(P_i);
7 S_s.W ← ω(P_i);
```

FIGURE 3.8: *A procedure for adding a new segment to the skyline during execution of the BL algorithm when an item has been packed in the left-hand corner of a skyline.*

FIGURE 3.9: *An example of an overhang.*

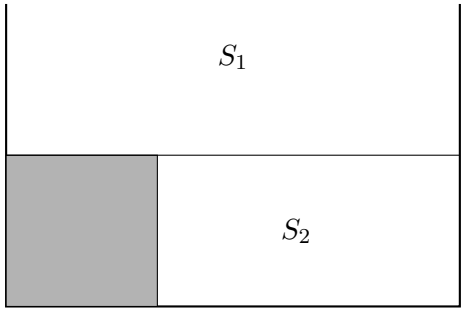
```
1  if i = 0 then
2      MoveSpace ← 0;
3      Exit;
4  MoveSpace ← W;
5  for j ← 1 to i do
6      if P_j.X + P_j.W ≤ P_i.X and P_j.Y < P_i.Y + h(P_i) and P_j.Y + h(P_j) > P_i.Y then
7          if MoveSpace > P_i.X − (P_j.X + P_j.W) then
8              MoveSpace ← P_i.X − (P_j.X + P_j.W);
9              if MoveSpace = 0 then
10                 Exit;
```

FIGURE 3.10: *A procedure for determining how far left an item $\mathcal{I}_i$ may be moved from its incumbent position during execution of the BL algorithm.*

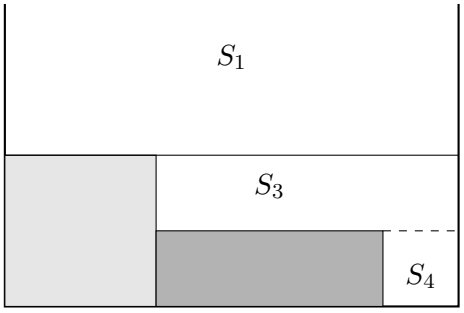
3.3 The Improved Heuristic Recursive

In 2006, Zhang *et al.* [165] proposed the fast new *heuristic recursive* (HR) algorithm for solving the two-dimensional SPP approximately. The HR algorithm is based on a *divide-and-conquer* strategy: It decomposes the original problem into several subproblems, conquers the subproblems by solving them recursively, and combines the solutions thus obtained to form a final packing solution. Items are initially sorted in order of non-increasing area. The first step of the algorithm consists of packing the first item in the bottom left-hand corner of the strip. The first level is thus defined and the remaining space of the strip is partitioned into two regions — the unoccupied area below the level boundary, referred to as the *bounded region*, and the space above the level boundary, referred to as the *unbounded region* (see Figure 3.11(a)). The unbounded region is

similar to the original strip and may be filled in the same manner as in the first step. The process of packing in the bounded region is achieved recursively by selecting the first unpacked item, placing it in that space (left-justified), and partitioning the remaining space into two bounded spaces (see Figure 3.11(b)). The filling process of all these bounded spaces within a level must be completed before proceeding to the next level, and the entire process is repeated until no unpacked items remain. The two procedures for packing the bounded and unbounded space, embedded in the HR strategy, may be found in Algorithms 3.3 and 3.4, respectively.



(a) *Partitioning a level defined by a packed item into an unbounded region S_1 and a bounded region S_2.*



(b) *Partitioning a bounded region into two subregions, S_3 and S_4, after an item is packed on it.*

FIGURE 3.11: *The process of partitioning the unpacked space into two subspaces by the IHR algorithm.*

Algorithm 3.3: **Packing procedure for the bounded space** (RecursivePacking())

```
1 initialise space stack, Sstack ← S_2;
2 while Sstack ≠ ∅ do
3     set current bounded space CurrBS to the uppermost element in Sstack and remove
      this element from Sstack;
4     pack the first item that fits into CurrBS;
5     divide the unpacked space into two bounded spaces S_3 and S_4;
6     add S_3 and S_4 to Sstack;
```

Algorithm 3.4: **Packing procedure for the unbounded space** (Packing())

```
1 initialise the unbounded space, S ← the strip;
2 while the packing process is not complete do
3     pack the first item into S;
4     divide the unpacked space into an unbounded space S_1 and a bounded space S_2;
5     S ← S_1;
6     RecursivePacking(S_2);
```

A year later, Zhang *et al.* [164] suggested the *improved heuristic recursive* (IHR) algorithm for the SPP in which 90 degree rotation of items is allowed. Because such rotation is beyond the scope of this dissertation, the IHR algorithm is described here for oriented SPPs only. The IHR procedure first combines all items of the same height to form a layer without wasted space, referred to as a *combination layer*. This search process is carried out repeatedly until all possible combination layers have been found. The algorithm, thereafter, packs the remaining items according the HR strategy. A pseudocode representation of the IHR algorithm is provided in Algorithm 3.5.

Algorithm 3.5: **Improved Heuristic Recursive algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.

Output: A feasible packing of the items in $\mathcal{P}$ into a strip of width W.

1 sort the list of items $\mathcal{P}$ according to non-increasing area;
2 form successive layers without wasted space by combining items of the same height;
3 pack the remaining items according to Packing();

Example 3 *The procedure for finding combination layers, when applied to the list of items in Table 3.1, yields only one combination layer, the first layer which contains items $\mathcal{I}_{14}$ and $\mathcal{I}_{16}$ shown in Figure 3.12. Items $\mathcal{I}_4$ and $\mathcal{I}_7$ have the same height, but these items do not form a combination layer, since the sum of their widths is less than the width of the strip. The same argument applies to items $\mathcal{I}_5$, $\mathcal{I}_6$, $\mathcal{I}_8$ and $\mathcal{I}_9$, and to items $\mathcal{I}_2$, $\mathcal{I}_3$, $\mathcal{I}_{10}$ and $\mathcal{I}_{11}$. Now, by sorting the remaining items in the list according to non-increasing area, the list $\mathcal{I}' = \{\mathcal{I}_4, \mathcal{I}_1, \mathcal{I}_7, \mathcal{I}_9, \mathcal{I}_5, \mathcal{I}_{12}, \mathcal{I}_{13}, \mathcal{I}_{15}, \mathcal{I}_8, \mathcal{I}_{11}, \mathcal{I}_2, \mathcal{I}_6, \mathcal{I}_{10}, \mathcal{I}_3\}$ results. The next item, $\mathcal{I}_4$, is placed on top of $\mathcal{I}_{14}$, left-justified. Two regions may now be defined: The bounded area between the right-hand edge of $\mathcal{I}_4$ and the right-hand boundary of the strip, and the unbounded area above the level boundary on top of $\mathcal{I}_4$.*

Items $\mathcal{I}_7$ and $\mathcal{I}_{12}$ are placed next to each other in the bounded area. No further items fit into this space, resulting in the initialisation of a new level by the item $\mathcal{I}_1$. This yields new bounded and unbounded spaces: The space between the right-hand edge of $\mathcal{I}_1$ and the right-hand boundary of the strip, and the area above $\mathcal{I}_1$, respectively. Items $\mathcal{I}_5$, $\mathcal{I}_6$, $\mathcal{I}_8$ and $\mathcal{I}_9$ have the same height and fit into the bounded region; they are therefore packed next to $\mathcal{I}_1$. The bounded space for packing is now the space above these items. The next item in the list, $\mathcal{I}_{13}$, is placed there, left-justified, resulting two new bounded spaces: To its right and above it. Items $\mathcal{I}_{15}$ and $\mathcal{I}_{10}$ are packed in the first bounded space, while items $\mathcal{I}_2$, $\mathcal{I}_3$ and $\mathcal{I}_{11}$ are placed in the second bounded space. The packing is shown graphically in Figure 3.12. ■

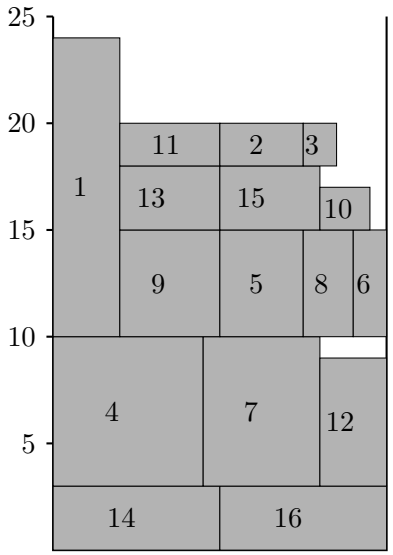


FIGURE 3.12: *Result obtained when packing items in $\mathcal{I}$ using the IHR algorithm. The resulting packing height is* $\mathtt{H} = 24$.

3.4 The Best-Fit Algorithm

The heuristics described above sort the list of items to be packed according to some criterion, such as by non-increasing height or area, and then apply a placement rule to pack items one by one in this order. The *best-fit* (BF) algorithm proposed by Burke *et al.* [29] is not, however, restricted to pack the next item encountered, but dynamically searches the list of unpacked items for the best-suited item for placement. The notion of a skyline representation, as discussed in some detail in §3.2, is employed to reduce the time required to find possible locations for unpacked items. Although the algorithm was originally created for packing problems allowing rotation, Ntene [128] showed how it can be applied to oriented strip packing.

The BF algorithm identifies the lowest location or gap in the strip, then finds the widest item that fits into it, and packs the selected item into the gap — in the left-most position, adjacent to the tallest neighbour, or adjacent to the shortest neighbour. The leftmost packing policy is utilised throughout this dissertation. If the width of the lowest gap found cannot accommodate any unpacked items, then it is raised to the height of its shortest neighbour. The entire process is repeated until all items are packed. A pseudocode representation of the BF algorithm is provided in Algorithm 3.6.

Algorithm 3.6: **Best-fit algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.
Output: A feasible packing of the items in $\mathcal{P}$ into a strip of width W.

```
1 sort the list of items in any manner;
2 initialise the skyline;
3 while there are unpacked items do
4     identify the lowest gap of the skyline;
5     if there is an item that fits into the space then
6         pack the first item that fits into the space;
7         update the skyline;
8     else
9         raise that skyline part to the height of the lowest of its neighbours;
```

Example 4 *By sorting the instance in Table 3.1 according to decreasing height, the list* $\mathcal{I}' = \{\mathcal{I}_1, \mathcal{I}_4, \mathcal{I}_7, \mathcal{I}_{12}, \mathcal{I}_5, \mathcal{I}_6, \mathcal{I}_8, \mathcal{I}_9, \mathcal{I}_{13}, \mathcal{I}_{14}, \mathcal{I}_{15}, \mathcal{I}_{16}, \mathcal{I}_2, \mathcal{I}_3, \mathcal{I}_{10}, \mathcal{I}_{11}\}$ *results. Initially there is only one skyline segment, the floor of the strip. The best item to fill this space is* $\mathcal{I}_1$ *and is packed into the bottom-left corner of the strip. This creates two skyline segments: The first is the part that spans the top edge of* $\mathcal{I}_1$ *and the second is the section between* $\mathcal{I}_1$ *and the right-hand boundary of the strip. The lowest skyline segment has a width of 16. The subsequent items,* $\mathcal{I}_4$ *and* $\mathcal{I}_7$*, have a total width of 16 and are therefore packed adjacent to* $\mathcal{I}_1$*, as shown in Figure 3.13. The packing now consists of two segments and the lowest segment is above* $\mathcal{I}_4$*. This segment has a width of 16 and is large enough to accommodate* $\mathcal{I}_{12}$*,* $\mathcal{I}_5$*,* $\mathcal{I}_6$*,* $\mathcal{I}_8$ *and* $\mathcal{I}_3$*.*

At this stage, the skyline consists of four segments: The top edges of $\mathcal{I}_1$*,* $\mathcal{I}_{12}$*,* $\mathcal{I}_3$*, and the part that spans the top edges of* $\mathcal{I}_5$*,* $\mathcal{I}_6$ *and* $\mathcal{I}_8$*. The lowest segment has width 2 and is narrower than each unpacked item. It is therefore raised to the same height as the skyline segment to its left. This segment now has a width of 12 and can accommodate* $\mathcal{I}_2$ *and* $\mathcal{I}_9$*. At this stage the packing yields a new skyline profile consisting of five segments in which the part along the top edge of*

$\mathcal{I}_3$ is the lowest segment. This segment is too narrow to accommodate the remaining unpacked items and it is thus raised to the height of the segment above $\mathcal{I}_9$. The segment above $\mathcal{I}_{12}$ is now the lowest and wide enough to contain $\mathcal{I}_{10}$. This yields a residual space too narrow for any of the unpacked items. Once the segment is raised to the height of the top edge of $\mathcal{I}_2$, item $\mathcal{I}_{13}$ may be packed into the resulting space.

The skyline is now reduced to three segments. The segment above $\mathcal{I}_1$ is the lowest, but narrower than any of the unpacked items. Raising the segment to the height of the segment above $\mathcal{I}_{10}$ yields a segment of width 7. Item $\mathcal{I}_{15}$ is the best item to fill this space and is packed left-justified into it. This yields a segment too narrow to contain any unpacked items; hence it is raised to the same height of the skyline segment to its right. Item $\mathcal{I}_{11}$ may be packed there; resulting in three new segments. The lowest segment cannot accommodate any of the unpacked items, hence it is raised to the height of its neighbouring segment. The same argument applies to the incumbent lowest segment. The packing now consists of one skyline segment — that along the top edge of $\mathcal{I}_{11}$ with a width of 20. Items $\mathcal{I}_{14}$ and $\mathcal{I}_{16}$ may be packed on that segment to yield a packing height of 22. The final packing is shown graphically in Figure 3.13. ■

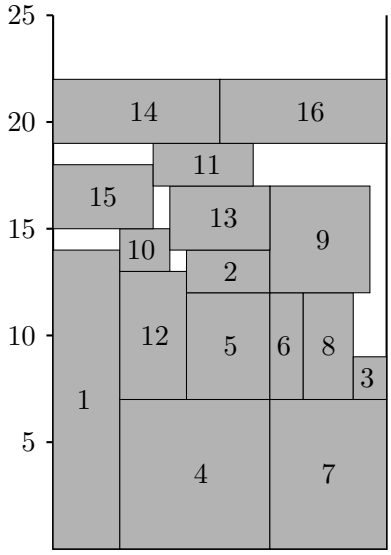


FIGURE 3.13: *Result obtained when packing items in $\mathcal{I}$ using the BF algorithm. The resulting packing height is* $\mathtt{H} = 22$.

The BF algorithm may be implemented efficiently by making use of the skyline representation described in §3.2. Once an item has been packed, the procedure in Figure 3.8 may be used to update the skyline, by adding a new segment to the skyline, in order to reflect the position of the packed item. It is important, however, to ensure that the skyline is updated accurately. It is not sufficient simply to add a new segment to the skyline after an item has been packed or simply to raise the skyline segment to the height of its lowest neighbour if it is too narrow to accommodate a particular item. If the new height of the skyline segment has the same height as any of its neighbours, then these segments should be combined and represented by one segment in the array. This prevents the algorithm from packing a narrow item into a narrow segment when the adjacent segments have the same height and could have accommodated a wider item. Therefore, after every skyline modification, the neighbours of the changing skyline segment are examined to determine whether their heights are the same (see Figure 3.14 for a description in pseudo-code form of a left-hand inspection procedure designed for this purpose).

```
1 S_s ← active skyline segment;
2 if S_{s−1} ← exists then
3     if S_{s−1}.Y = S_s.Y then
4         S_{s−1}.W ← S_{s−1}.W + S_s.W;
5         S_{s−1} ← active skyline segment;
6         S_s ← delete;
```

FIGURE 3.14: *A procedure for combining adjacent segments of a skyline on the left-hand side during execution of the BF algorithm.*

3.5 The Constructive Heuristic

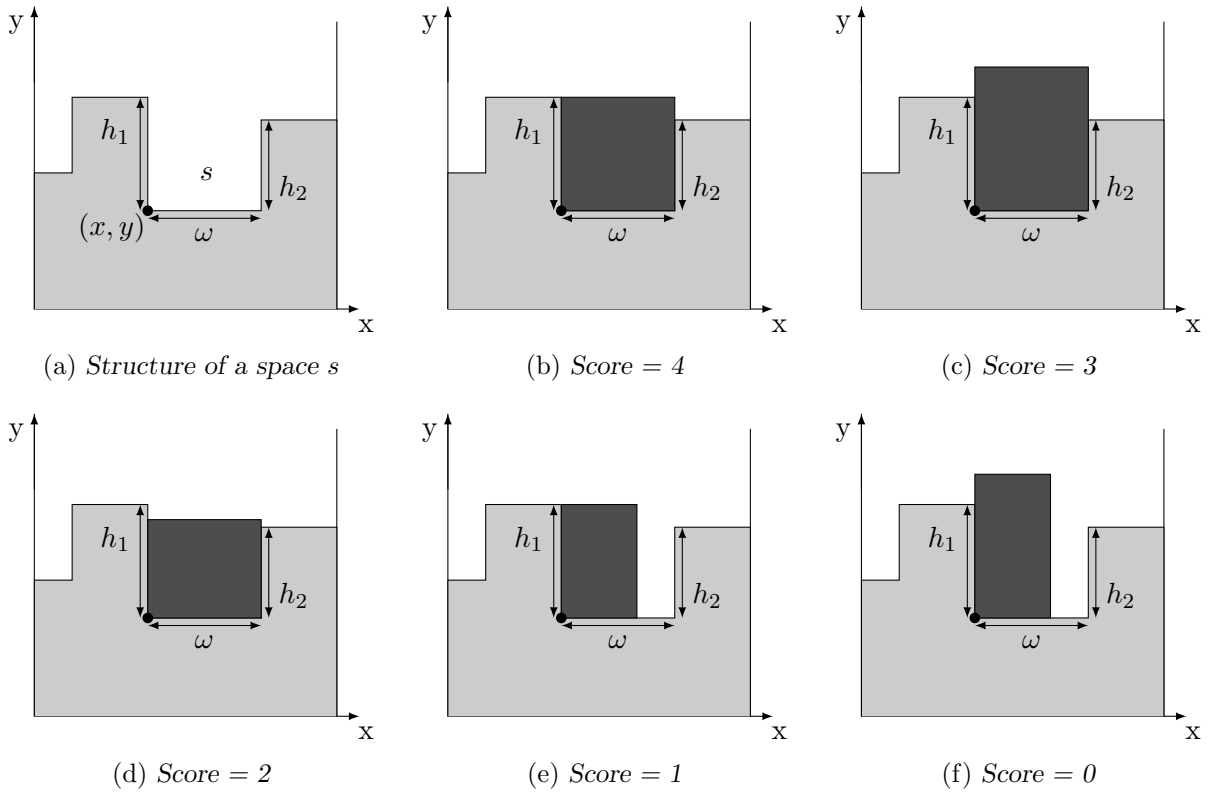
In 2011, Leung and Zhang [115] proposed the *fast-layer heuristic* (FH) for the non-guillotine SPP, by allowing rotation of the items. It is a layer-based algorithm attempting to fill each layer by following bottom-left and stacking strategies. In particular, the FH algorithm makes use of an evaluation measure in order to select the best item to pack. In this dissertation, an alternative version of the algorithm, called the *constructive heuristic* (CH) proposed by Leung *et al.* [116] during the same year, is adopted. The CH has subsequently been embedded in a two-stage intelligent search algorithm and is appropriate for the oriented SPP.

The CH algorithm is based on a scoring rule which facilitates the identification of the best item to pack in the currently available space. The notion of a skyline, as proposed by Burke *et al.* [29], is adopted to define all available spaces, where each space is associated with five variables, namely its position (its bottom left-hand corner coordinates x and y), its width ω, the height of the left wall h_1 and the height of the right wall h_2, as illustrated in Figure 3.15(a). The CH algorithm finds the lowest and left-most available space, and scores each unpacked item for that space. The item with the highest score value is selected and is packed there adjacent to the tallest neighbour. The scoring rule utilised by Leung *et al.* is summarised in Table 3.2 and is illustrated by means of an example in Figure 3.15(b)–(f). The process continues until no unpacked items remain. A pseudocode listing of the CH algorithm is provided in Algorithm 3.7.

If	Conditions	Score
$h_1 \geq h_2$	ω = item.width and h_1 = item.height	4
	ω = item.width and $h_1 <$ item.height	3
	ω = item.width and $h_1 >$ item.height	2
	$\omega >$ item.width and h_1 = item.height	1
	$\omega >$ item.width and $h_1 \neq$ item.height	0
$h_1 < h_2$	ω = item.width and h_2 = item.height	4
	ω = item.width and $h_2 <$ item.height	3
	ω = item.width and $h_2 >$ item.height	2
	$\omega >$ item.width and h_2 = item.height	1
	$\omega >$ item.width and $h_2 \neq$ item.height	0

TABLE 3.2: *The scoring rule utilised by Leung* et al. *[116] for determining the best item that fits into a selected available space s. The parameters h_1, h_2, and ω are the height of the left wall, the height of the right wall and the width of s, respectively.*

Example 5 *By sorting the instance in Table 3.1 according to non-increasing height and resolving ties (items of equal height) by additionally sorting those items by decreasing width, the list* $\mathcal{I} = \{\mathcal{I}_1, \mathcal{I}_4, \mathcal{I}_7, \mathcal{I}_{12}, \mathcal{I}_9, \mathcal{I}_5, \mathcal{I}_8, \mathcal{I}_6, \mathcal{I}_{14}, \mathcal{I}_{16}, \mathcal{I}_{13}, \mathcal{I}_{15}, \mathcal{I}_{11}, \mathcal{I}_2, \mathcal{I}_{10}, \mathcal{I}_3\}$ *results. The initial available space for packing is the strip with a width of 20. According to the scoring rule,*



(a) *Structure of a space s* (b) *Score = 4* (c) *Score = 3*

(d) *Score = 2* (e) *Score = 1* (f) *Score = 0*

FIGURE 3.15: *(a) Different variables associated with a space s. (b)–(f) Examples of the scoring rule used by Leung* et al. *[116] (for $h_1 \geq h_2$).*

all items in the list have a score equal to 0 with respect to that space. The best item to fill this space is thus $\mathcal{I}_1$ (the tallest item) and this item is packed into the bottom-left corner of the strip. This yields two new available spaces: The area above $\mathcal{I}_1$ and the space spanning the length of the floor of the strip not under $\mathcal{I}_1$. The latter space is the lowest space and item $\mathcal{I}_4$ is the best item that fits into it. Item $\mathcal{I}_4$ is, therefore, packed into that space right-justified, as shown in Figure 3.16, since the height of the right-hand boundary of the strip is larger than the height of $\mathcal{I}_1$. The packing now consists of three available spaces and the lowest of these is the region between $\mathcal{I}_1$ and $\mathcal{I}_4$. Item $\mathcal{I}_7$ is the only unpacked item that has the same width as the width of the lowest space; hence it yields the highest score with respect to that space and is packed there.

The number of available spaces is now reduced to two. The region above $\mathcal{I}_7$ and $\mathcal{I}_4$ is the lowest and is wider than any unpacked items. Hence $\mathcal{I}_{12}$, the tallest item among the remaining items, is the best item to fill that space and is packed right-justified into it. This creates a new space, the region between the left-hand edge of $\mathcal{I}_{12}$ and the right-hand edge of $\mathcal{I}_1$. This new space is the lowest available space for packing and is large enough to accommodate any unpacked items in the list. Items $\mathcal{I}_9$ and $\mathcal{I}_{13}$ are the best pair of items that fill this space and are packed into it. The region above $\mathcal{I}_{13}$ is now the lowest available space which has a width of 6 and a height of 3 on the right. Item $\mathcal{I}_{15}$ has the same width and height as this space, and is thus the best-suited item for that space. The same argument applies for the placement of $\mathcal{I}_{11}$ in the space above $\mathcal{I}_9$.

At this stage, the packing consists of two available spaces. The first space is the region above $\mathcal{I}_1$ and $\mathcal{I}_{11}$, while the other is the part along the top of $\mathcal{I}_{15}$ and $\mathcal{I}_{12}$. The second space is the lowest available position and has a width of 10. Item $\mathcal{I}_{14}$ is the next item in the list which has a large fitness value for that space, resulting in its packing there. The same applies for $\mathcal{I}_{16}$, which is the best-suited item for the space above $\mathcal{I}_1$ and $\mathcal{I}_{11}$, resulting in its placement there. The two spaces

Algorithm 3.7: **Constructive Heuristic algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.

Output: A feasible packing of the items in $\mathcal{P}$ into a strip of width W, and the packing height h.

```
1  initialise the skyline;
2  h ← 0, NumPackedItems ← 0;
3  while there are unpacked items do
4      find the lowest and the left-most space s of the skyline;
5      if there is an item that fits into s then
6          for each unpacked item i do
7              s_i = score(i, h_1(s), h_2(s), ω(s));
8          select the first item R with the maximum score;
9          if y + h(R) > h then
10             h ← y + h(R);
11         if h_1(s) ≥ h_2(s) then
12             pack R against the left wall;
13             update the skyline;
14         else
15             pack R against the right wall;
16             update the skyline;
17     else
18         update the skyline;
19 return h;
```

for packing are now defined as the region above $\mathcal{I}_{14}$, which is the lowest, and the part along the top of $\mathcal{I}_{16}$. Items $\mathcal{I}_2$ and $\mathcal{I}_5$ are the best pair of items that fill the lowest space and are packed into it. Similarly, items $\mathcal{I}_8$, $\mathcal{I}_6$, $\mathcal{I}_{10}$ and $\mathcal{I}_3$ fit perfectly into the area above $\mathcal{I}_{16}$ and are packed there. The final packing layout is shown in Figure 3.16. ■

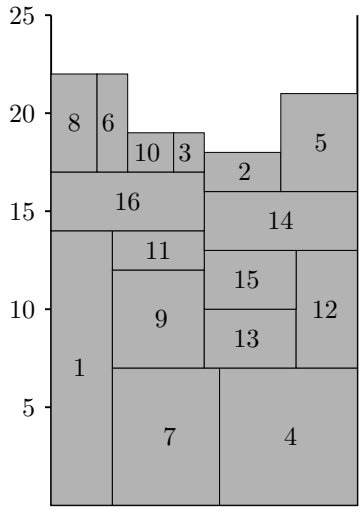


FIGURE 3.16: *Result obtained when packing items in $\mathcal{I}$ using the CH algorithm. The resulting packing height is* $\mathtt{H} = 22$.

3.6 Chapter Summary

Five existing heuristics for the oriented 2D SPP were reviewed in this chapter. The BFDH* algorithm by Bortfeldt [25] was discussed first in §3.1, and this was followed by a description of an improved version of the BL algorithm proposed by Liu and Teng [118] for use in their hybrid genetic algorithm in §3.2. Thereafter, the IHR of Zhang *et al.* [165] was reviewed in §3.3, and this was followed by a review of the BF algorithm of Burke *et al.* [29] in §3.4. The CH algorithm of Leung *et al.* [116] was finally discussed in some detail in §3.5.

CHAPTER 4

Strip Packing Metaheuristics

Contents

Various metaheuristic algorithms developed for the 2D SPP are described in this chapter. These algorithms are mainly hybrid approaches, combining metaheuristic search techniques with heuristic algorithms. The basic working of metaheuristic search techniques are first discussed in §4.1 in the context of packing problems. More specifically, the problem-specific and generic design variables involved in genetic algorithms are presented in §4.1.1, and this is followed by a brief description of the working of simulated annealing in §4.1.2. A representative sample of known strip packing metaheuristics is described thereafter in some detail in §4.2. A summary of the chapter contents is finally presented in §4.3.

4.1 Two Popular General Metaheuristic Search Techniques

In order to overcome the main disadvantage of heuristic packing techniques, such as those described in Chapter 3, whose main weakness lies in a general inability to provide good solution quality, superior and more effective heuristic search strategies have been developed in the strip packing literature. These search strategies include the well-known class of metaheuristic techniques, which operate at a higher level of flexibility than heuristics and typically provide sufficiently near-optimal solutions. Various metaheuristic search principles have been proposed during the last four decades. Some of these have been inspired by nature and are modelled on processes such as biological evolution and physical annealing of solids. Genetic algorithms and

simulated annealing are the most widely applied metaheuristics for packing problems [31, 89, 91, 113]. The basic working of a generic GA and the method of SA are presented in this section.

4.1.1 Genetic Algorithms

GAs, originally proposed by Holland in 1975 [87], are search procedures that operate in a similar way to evolutionary processes observed in nature. These algorithms attempt to find better solutions to an optimisation problem instance by iteratively improving a set of current candidate solutions. The search is guided towards improvement by applying the principle of "survival of the fittest" — the most desirable features from each generation of solutions are extracted and combined to form the next generation. The quality of each solution is measured by a fitness function, which essentially depends on the objective function associated with the problem instance under consideration. The various problem-specific and generic design variables embedded in GAs include a *solution representation technique*, an *initial solution population*, a *fitness function*, various *genetic operators* (selection, crossover, mutation), and a *termination criterion*.

The solution representation technique employed to encode solutions is important for the successful operation of GAs. A good representation of candidate solutions of the optimisation problem instance under consideration reduces algorithmic effort during the process of decoding. The classical approach involves binary coding where each candidate solution is represented by strings of zeros and ones [35]. In the context of packing problems, the representation of a solution or a packing pattern, as a data structure for encoding solutions in a GA, takes the form of permutation $\pi = (i_1, i_2, \ldots, i_n)$, where $i_j \in \{1, 2, \ldots, n\}$ denotes the index of the j-th item packed and n is the total number of items in the given instance [99]. The permutation therefore represents the order in which the items are to be packed.

The initial population is the first set of solutions generated at the beginning of the search. Usually, the initial population is generated randomly. Alternatively, it can be seeded with high-quality solutions in an attempt to help the GA to find better solutions. The size of the population is usually fixed.

Each individual in a population is associated with a fitness level which specifies the desirability of the individual being selected for reproduction or replacement. Fitness levels are determined by a fitness function which depends on the objective function associated with the optimisation problem at hand. In the context of the SPP, the fitness function is usually determined by the height of the packing pattern [93].

During the reproduction process, candidate solutions are selected from the current population and copied into a mating pool for production purposes in order to form the next generation. It is a probabilistic process in which the selection probability of an individual depends on its fitness. Fitter individuals are more likely to be selected while unfit solutions are less likely to partake in reproduction. A distinction is made between two types of selection operators: The *selection for reproduction*, which determines the likelihood that a candidate solution will be chosen to reproduce, and the *selection for replacement*, which determines the specific individuals that will survive to the next generation.

Proportional selection [87] is the most popular type of selection rule for reproduction in the context of the SPP. The probability of selection of an individual is proportional to its fitness. This technique can be implemented in different ways. The simplest method is known as *roulette wheel* selection. Each individual is allocated a circular section resembling that on a roulette wheel, arc-sized in proportion to its fitness. The probability of selection of each individual is

proportional to the area of its corresponding slot on the roulette wheel. Random numbers are generated one at a time according to a uniform distribution in order to select individuals for parenthood. An alternative method is the so-called *stochastic universal selection* (SUS) method [12]. The method of roulette wheel selection exhibits a single choice at each application (hence it is required to spin the roulette wheel as many times as the number of individuals that have to be selected for parenthood), while the SUS selection generates the required number of parent selections simultaneously by choosing them at evenly spaced intervals along the wheel.

Other selection strategies for reproduction include *ranking selection* and *tournament selection* [68]. In cases where the fitness values of the members of a population of candidate solutions differ significantly, the fittest individuals may dominate the recombination process while the other individuals may have very small probabilities of being selected. This may lead to a reduction of genetic diversity during the algorithmic execution process. Ranking selection may be applied to avoid this shortcoming. After sorting the individuals according to their fitness values, reproductive fitness values may be assigned according to the ranks of individuals, either linearly [159] or exponentially, as discussed in [23]. A parent is thus selected based on its rank rather than based on its absolute fitness value. Tournament selection, on the other hand, consists of choosing and subsequently comparing a set of individuals from the population, and returning the best individual from the selected subset for parenthood.

At the end of the reproduction process, the incumbents and the newly produced candidate solutions are subject to a replacement process aimed at maintaining a fixed population size. Two main replacement techniques prevail in the literature. The first is known as *generational replacement*, according to which the entire population is replaced at once, while the second is known as *steady state replacement*, according to which only one (or few) member(s) of the generation is (are) replaced at a time [68, 147].

Once the mating pool has been created, *crossover* and *mutation operators* are applied in order to produce new offspring. During crossover, features of the parents selected are combined to generate one or more offspring solutions. The rationale behind crossover is to transfer good solution features to the next generation of candidate solutions in order to achieve progressively better solutions over time. The following crossover operators are suitable for the type of encoding scheme employed in the context of the SPP:

***Partially Matched Crossover* (PMX).** This type of crossover was developed by Goldberg and Lingle [70]. According to their approach, two cut points are chosen uniformly at random positions along the encodings of parent solutions. Offspring solutions are then created in such a way that the first offspring solution receives a copy of the substring components between the two crossover points from the second parent, and likewise for the second offspring and the first parent. In the following two parent solutions, for instance, the two crossover points are located between the third and fourth components, as indicated by the vertical divides, as well as between the sixth and seventh components, resulting in the generation of the two partial offspring solutions:

Parent 1:	[3 2 1 | 4 5 6 | 8 7]	Partial offspring 1:	[$\cdots$ | 1 6 8 | $\cdot$ $\cdot$]
Parent 2:	[3 7 5 | 1 6 8 | 2 4]	Partial offspring 2:	[$\cdots$ | 4 5 6 | $\cdot$ $\cdot$].

The section between the two cut points defines an interchange mapping. In the example above, the interchange mapping is $4 \leftrightarrow 1, 5 \leftrightarrow 6$ and $6 \leftrightarrow 8$. The remaining components in offspring solution $i \in \{1, 2\}$ are populated by copying the components of the i-th parent into the respective positions. In case a duplication occurs during this transfer,

the interchange mapping defined between the two cut points may be employed to replace the duplicate values. In the example above, the third component of Parent 1, namely the value 1, is already present in the partial offspring solution 1. The value 4 is thus placed in the third component of the offspring solution 1 instead of the value 1, by utilising the interchange mapping defined earlier. The same applies to other duplicate values that might occur during the transfer. According to this approach, the new offspring solutions of the above example are

$$\begin{array}{ll} \text{Offspring 1:} & [3\ 2\ 4 \mid 1\ 6\ 8 \mid 5\ 7], \\ \text{Offspring 2:} & [3\ 7\ 8 \mid 4\ 5\ 6 \mid 2\ 1]. \end{array}$$

***Position-based Crossover* (POS).** This type of crossover was proposed by Syswerda [148]. The procedure is initialised by selecting a random set of positions in the parent solutions, and copying the corresponding components from the first parent into the second offspring solution, and similarly for the second parent and the first offspring solution. Consider again the above parent solutions, and suppose that the second, third and sixth positions have been selected. This leads to the following offspring solutions:

$$\begin{array}{ll} \text{Offspring 1:} & [\cdot\ 7\ 5 \cdot\cdot 8 \cdot \cdot], \\ \text{Offspring 2:} & [\cdot\ 2\ 1 \cdot\cdot 6 \cdot \cdot]. \end{array}$$

The remaining components in offspring solution $i \in \{1, 2\}$ are finally inserted from left to right by copying the remaining components of the i-th parent in the same order that they appear in the solution encoding of that parent. The new offspring solutions are therefore:

$$\begin{array}{ll} \text{Offspring 1:} & [3\ 7\ 5\ 2\ 1\ 8\ 4\ 6], \\ \text{Offspring 2:} & [3\ 2\ 1\ 7\ 5\ 6\ 1\ 4]. \end{array}$$

***Cycle Crossover* (CX).** This type of crossover was proposed by Oliver *et al.* [130]. This operator attempts to create offspring solutions by generating successive cycles from parent solutions and copying the related values into the offspring solutions until they are fully constructed. According to this approach, an offspring solution is composed of a series of alternating cycles, *i.e.* if the first copied cycle is from the first parent, then the next cycle is obtained from the second parent, and so on. A cycle from a particular parent may be created by initialising it with the first component of that parent and then repeatedly adding the corresponding value of its previous component from the other parent until the first element of the cycle is reached again. An illustrative example of the CX operator is presented in Figure 4.1.

Mutation operators modify the solution encodings of offspring solutions slightly to form alternative solutions. The rationale behind mutation is to facilitate escaping from local optima and hence to prevent the search process from converging prematurely on a locally optimal solution. Different methods of applying mutation exist in the literature [146]. The most commonly adopted approach in the context of SPP is the so-called *swap operator*: Two components in a solution encoding are selected at random according to this approach, and their values are exchanged to yield a new solution encoding. In the following example, the third and sixth components are selected in order to affect crossover, which results the solution shown on the right:

$$[1\ 2\ \mathbf{3}\ 4\ 5\ \mathbf{6}\ 7\ 8] \quad \Rightarrow \quad [1\ 2\ 6\ 4\ 5\ 3\ 7\ 8].$$

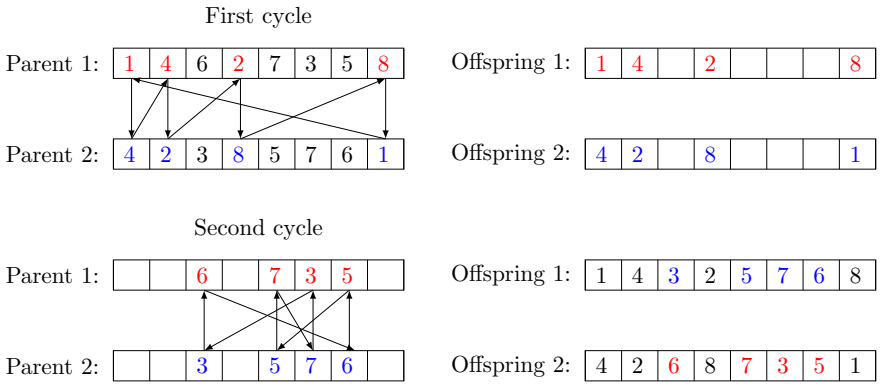


FIGURE 4.1: *Example of the working of the CX crossover operator.*

Since GAs are iterative search techniques which offer near-optimal populations of solutions if implemented suitably, a criterion for the termination of the search process is required. This can be achieved by fixing the number of iterations for the entire search process. An alternative approach is to terminate the process when no further improvements have been observed over a specified number of generations.

4.1.2 Simulated Annealing

The method of SA is an iterative search process due to Kirtkpatrick *et al.* [109] that operates in a manner analogously to the process of physical annealing of solids, in which a physical system is led to a low energy state by carefully controlling its temperature. A high energy state of a solid is achieved by initially heating it and then allowing it to cool slowly in stages until a stable solid state with a minimum energy configuration is obtained. Transferring this analogy to the context of optimisation problems, the energy states correspond to the various feasible solutions while the energy of the system is related to the objective function to be optimised.

The key feature of SA is that it provides a means to escape from local optima by allowing *hill-climbing* moves, *i.e.* moves along which the objective function value worsens. Instead of only accepting neighbouring solutions that result in an improvement of the objective function, inferior solutions may also be accepted with a certain probability. This probability depends on the magnitude of the degradation in the objective function value and an external control parameter, called the *temperature*, which is gradually lowered according to a so-called cooling schedule. The temperature is controlled in such a manner that the algorithm becomes more selective in accepting new solutions, and accepts predominantly improving solutions toward the end of the search process.

The basic problem-specific and generic parameter variables involved in SA include an *initial solution*, a *neighbourhood structure*, an *initial temperature*, a *cooling schedule* (a mathematical expression describing how the temperature is lowered), an *appropriate length of epoch management rule* (the number of iterations over which the temperature is held constant during the search), and a *termination criterion* [32, 57, 152].

The initial solution is the first solution considered at the beginning of the search. In contrast to a GA search, which requires an initial population of solutions, only one candidate solution is generated at random in the case of the SA procedure.

Associated with a particular solution are neighbouring solutions that can each be reached in a single move from the current solution being considered. These neighbouring solutions are

generated according to a pre-specified rule. In the context of the SPP, the neighbours of a solution encoding are typically the set of permutations obtained by reversing the order of any two randomly selected items in the permutation [31, 91, 116].

The SA cooling schedule typically consists of three parameters: an initial temperature, a cooling rule, and a number of move transitions for each temperature value [32, 88]. No specific rule has been defined that provides the best cooling schedule and initial temperature value combination for all optimisation problems — these are determined empirically in the context of the particular optimisation problem that has to be solved.

Generally, the initial value of the temperature parameter is selected high enough that the solution space is sufficiently explored during the early stages of the search (*i.e.* any new solution is accepted with a probability close to 1 at the beginning of the search). During the search, the temperature is held constant for a pre-specified number of iterations or an epoch. The duration of an epoch may be determined dynamically so as to promote a high level of flexibility. In [91], for example, an epoch is terminated when $50N$ moves or $5N$ successful moves have been attempted, where N denotes the number of items in an SPP instance.

The most commonly adopted rule for reducing the temperature in the SPP literature is the *geometric schedule* [31, 91, 116]. According to this schedule, the new temperature is obtained by multiplying the incumbent temperature by a so-called *cooling parameter* (smaller than 1).

The search process typically terminates when a pre-specified termination criterion is satisfied. An example is when a maximum number of iterations has been performed [31]. An alternative termination criterion is when the search arrives at a lower bound on temperature [116] or when no improvement in the solution is achieved over a pre-specified number of successive search iterations [91].

4.2 Strip Packing Metaheuristics

In this section, a representative sample of known strip packing metaheuristics is reviewed in some detail. A brief description of each algorithm is followed by a pseudocode listing of the procedure, together with a packing result returned by the algorithm (when applied to the instance $\mathcal{I}$ in Figure 3.1) in each case.

4.2.1 Hybrid Genetic Algorithms

As discussed in §2.2.3, three classes of strip packing solution approaches involving GAs exist in the literature. The methods in the first class employ codings of solutions and therefore apply decoding routines to transform the encoded solutions into complete SPP layouts. Approaches in the second class employ problem-specific encodings and corresponding problem-specific operators to solve the SPP. Solution approaches in the third class do not use any encoding of solutions; these methods rather solve the problem instance directly in the actual packing space. Hybrid GAs belong to the first class and are typically two-stage approaches in which a GA is combined with a heuristic placement policy. The task of the GA is to search for a good packing order in which to pack items, while the heuristic routine is used to evaluate the encoded solutions (*i.e.* permutations) by transforming these encodings to corresponding physical SPP layouts.

The working of a hybrid GA may be described as follows. The algorithm first generates an initial population, which is typically composed of randomly generated packing permutations. The initial population can alternatively be seeded with high-quality solutions, such as packing

permutations resulting from sorting items according to decreasing height or width. The heuristic placement routine is then used to evaluate the quality of each packing permutation in the population according to a fitness function. The quality of a packing permutation is typically determined by the packing height of its corresponding SPP layout according to some heuristic packing scheme. Alternative fitness functions are the weighted sum of the packing height and the packing density, as suggested by Hopper and Turton [91], or the area of contiguous remainder[1] associated with a packing pattern, as proposed by Jakobs [99].

A mating pool of solutions for reproduction purposes is then created from the current population according to an appropriate selection criterion. Proportional selection, based on the roulette wheel method, has been used by Hopper and Turton [91] and by Jakobs [99] in their hybrid GA implementations. As described in §4.1.1, each individual is allocated a roulette wheel slot sized in proportion to its fitness. The probability of selection of each individual is proportional to the area of its corresponding slot on the roulette. Random numbers are generated according to a uniform distribution, one at a time, in order to select individuals for parenthood. Upon execution of this process, crossover and mutation operators are applied to the mating pool in order to generate a new generation of candidate solutions. PMX crossover has been employed by Hopper and Turton [91], Jakobs [99], Liu and Teng [118], and Zhang *et al.* [164] in their hybrid GA implementations.

The mutation operator is applied after the crossover operation. Exchange or swap mutation is the most commonly adopted mutation operator in the context of hybrid GAs for the SPP. As explained in §4.1.1, this technique involves swapping two positions of items in the packing permutation at random. After applying this operation, the fitness values of the offspring solutions generated are evaluated by means of the heuristic placement routine. At this stage, a selection for replacement strategy is applied to both the previous population of candidate solutions and the offspring population of candidate solutions in order to update the current population. An elitism strategy has been employed by Hopper and Turton [91], Liu and Teng [118], and Zhang *et al.* [164] in their hybrid GA implementations, while Jakobs [99] adopted steady-state replacement in his approach. The best packing permutation found is saved and the process starting from the creation of a mating pool for the new population is repeated until a stopping criterion is satisfied. A pseudocode listing of a hybrid GA combined with the BL algorithm as decoding mechanism may be found in Algorithm 4.1.

Example 6 *An example of a solution obtained by the hybrid GA combined with the BL algorithm decoding mechanism, when applied to the instance $\mathcal{I}$ of Figure 3.1, is given in Figure 4.2. In this example, the population size was taken as 50, the crossover probability was selected as 90% and the mutation probability as 10%. The quality of a packing solution was determined by its packing height. SUS selection, PMX crossover and elitism replacement operators were implemented. The initial population was generated randomly and the entire search was terminated after 1 000 generations. The solution depicted in Figure 4.2 was the best solution encountered during the search, and occurred as a member of the 640-th generation.* ■

4.2.2 Hybrid Simulated Annealing

The application of the method of SA to the SPP is mainly based on a hybrid approach in which SA is combined with a heuristic placement as decoding routine. Similar to hybrid GAs

[1] The area of contiguous remainder associated with a packing pattern is the remaining area above the skyline delimited by the items packed last and a pre-defined height of the strip.

Algorithm 4.1: **Hybrid Genetic Algorithm with Bottom-left algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.

Output: A feasible packing of the items in $\mathcal{P}$ into a strip of width W.

```
generate an initial population G_0 of size N;
X_1 ← permutation sorted according to decreasing width;
Fitness(X_1) ← BL(X_1);
for i ← 2 to N do
    X_i ← random permutation;
    Fitness(X_i) ← BL(X_i);
while the stopping criterion is not yet satisfied do
    create a mating pool of solutions from G_t by a selection process based on Fitness;
    apply a crossover operator (according to a crossover probability p_c) and a mutation
    operator (according to a mutation probability p_m) to the mating pool;
    generate the next population G_{t+1} of size N using a replacement strategy;
    save the best permutation uncovered so far;
```

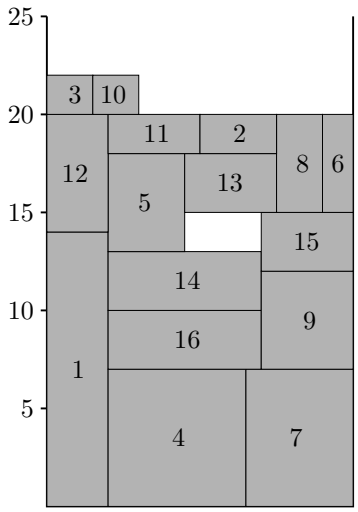


FIGURE 4.2: *An example of a packing solution obtained when applying the hybrid GA combined with the BL algorithm as decoding routine to the list of items $\mathcal{I}$ in Figure 3.1. The resulting packing height is* $\mathtt{H} = 22$.

described in §4.2.1, the task of the SA algorithm is to search for a good order in which items should be packed, while the heuristic decoding routine is used to evaluate the quality of a packing permutation by converting it into an actual packing layout.

A hybrid SA algorithm starts by generating an initial solution (*i.e.* a packing permutation) at random, and selecting an initial value of the temperature. It then applies the heuristic routine to evaluate the quality of the initial packing permutation. The algorithm further executes the following steps repeatedly until a pre-determined termination condition is satisfied. In the first step, a manipulation operator is applied to the current solution in order to generate a new neighbouring solution. The quality of this new solution is then evaluated by means of the heuristic placement routine. Upon execution of this process, the quality of the current solution is compared with that of the best solution found so far. If the current solution achieves a better packing layout than the best one, it becomes the new current solution; otherwise, it is rejected

unless the probability condition of accepting a worsening solution is satisfied. The value of the temperature is updated after a given epoch length. A pseudocode representation of a hybrid SA combined with the BL algorithm as decoding routine is shown in Algorithm 4.2.

Algorithm 4.2: **Hybrid Simulated Annealing with Bottom-left algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.
Output: A feasible packing of the items in $\mathcal{P}$ into a strip of width W.

```
1  generate an initial solution X_0, X ← X_0 (the current ordering);
2  generate an initial temperature T_0, T ← T_0 (the current value of the temperature);
3  BestSolution ← BL(X);
4  while the stopping criterion is not yet satisfied do
5      for a specific length of epoch do
6          define the neighbourhood structure of the current solution by means of the
           manipulation operator;
7          obtain a new neighbouring solution X';
8          CurrentSolution ← BL(X');
9          if CurrentSolution is better than BestSolution then
10             BestSolution ← CurrentSolution;
11         else
12             if the probability condition of accepting a worsening solution is satisfied then
13                 BestSolution ← CurrentSolution;
14     Update the temperature;
```

Example 7 *An example of a solution obtained by the hybrid SA combined with the BL algorithm as decoding routine, when applied to the instance $\mathcal{I}$ of Figure 3.1, is given in Figure 4.3. In this example, the cooling schedule was taken as the geometric schedule with a cooling rate of 0.93. The initial value of the temperature was taken as 0.5. The temperature was held constant for 16 moves (the total number of items in the SPP instance). The initial solution was randomly generated and the fitness function was taken as the packing height associated with a solution. The algorithm was run for 1 000 iterations. The packing solution illustrated in Figure 4.3 was the best solution obtained during the search, and occurred during the 953-th iteration.* ■

4.2.3 The SPGAL Algorithm

In 2006, Bortfeldt [25] proposed a GA solution approach for the 2D SPP which operates directly on the SPP search space. His SPGAL algorithm is an adaptation of the CLP-GA genetic algorithm for the container loading problem of Bortfeldt and Gehring [26]. According to the SPGAL algorithmic approach, multiple instances of the two-dimensional container loading problem are solved iteratively until a smallest length of the container, equivalent to a smallest packing height of a two-dimensional strip, is reached. Initially, the 2D SPP instance is solved by means of the BFDH* algorithm, as described in §3.1. The packing height thus obtained is used as the length of the container of the first two-dimensional container loading problem instance. This instance is solved by means of the CLP-GA, which provides a new solution to the 2D SPP instance. The container length is updated by reducing it by one unit, and the process is repeated until no further improvement is achieved (*i.e.* if the solution obtained does not contain all items). In

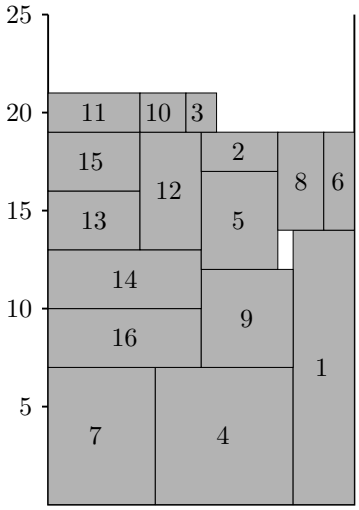


FIGURE 4.3: *An example of a packing solution obtained when applying the hybrid SA combined with the BL algorithm as decoding routine to the list of items $\mathcal{I}$ in Figure 3.1. The resulting packing height is* $\mathtt{H} = 21$.

order to fully understand the working of the SPGAL algorithm, the original CLP-GA for the container loading problem is first considered.

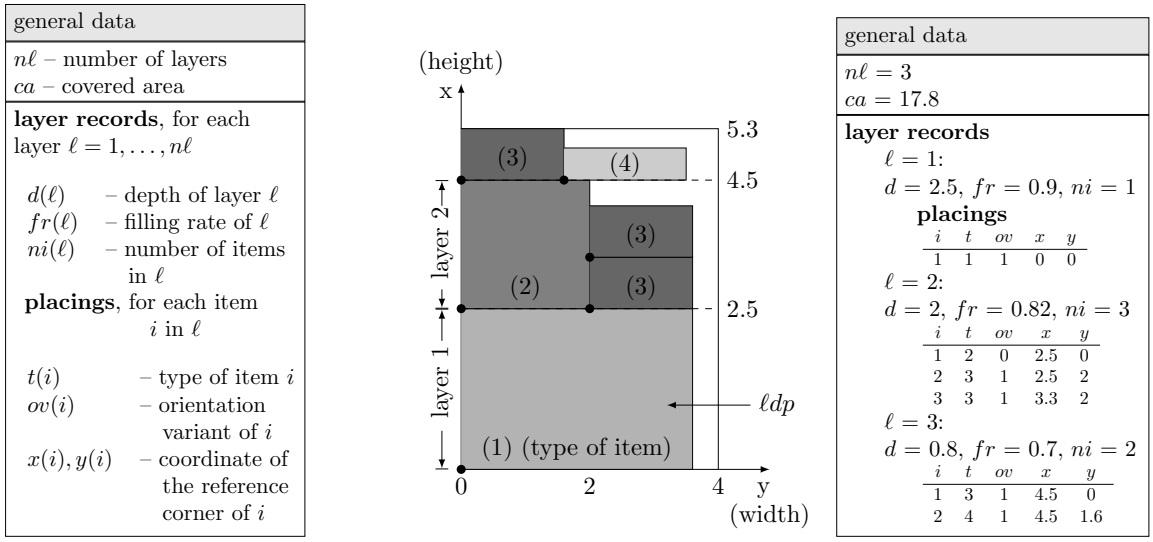
The CLP-GA for the Container Loading Problem

The CLP-GA for the two-dimensional container loading problem is implemented directly in the space of completely defined packing layouts according to a layer structure. Such a layout consists of successive rectangular layers in which one or more items are arranged. The width of a layer corresponds to the width of the container, while the layer depth is specified by a so-called *layer-determining piece* (*ldp*) and its orientation. The representation of a layout, as a data structure for encoding solutions in the GA, comprises the following variables: the *total number of layers*, the *covered area* (the total area occupied by all items placed), the *layer records* (the depth, the filling rate[2], and the number of items packed in each layer), and the *placings within a layer* (the type, the orientation, and the coordinates of the reference corner[3] of each item packed in a particular layer). The layers of a solution are always arranged in ascending order according to the filling rate. This type of layout representation is illustrated by means of an example in Figure 4.4.

The overall procedure of the CLP-GA is shown in pseudocode form in Algorithm 4.3. Taking as input the container dimensions and the list of items to be packed, the GA search starts by initialising a set of feasible layouts. Then, during each iteration, a subsequent generation is created by first reproducing the best *nrep* solutions of the previous generation and completing the population up to the full size by means of crossover and mutation operators. The fitness function consists of the covered part of the container area and the selection process is based on a ranking strategy. The crossover and mutation operators are performed alternatively according to constant probabilities. During the course of a crossover operation, all the layers of selected parents are examined in descending order according to the filling rate. The best parent layer is

[2]The filling rate of a layer is the ratio of the total area occupied by all items packed in a layer to the area of the layer.

[3]The reference corner of an item is the corner that is nearest to the origin of the two-dimensional coordinate system.



(a) *Data structure* (b) *Example of a layout and its corresponding solution encoding*

FIGURE 4.4: *(a) A layout representation as a data structure for encoding solutions in the CLP-GA. (b) A concrete example of a packing instance.*

always transferred to the new offspring solution. The next best parent layer is only transferred if its depth does not exceed the residual container length (the area defined by the boundary of the container and the last layer defined) and if the total number of existing items per type is respected (see Figure 4.5).

Algorithm 4.3: **CLP-GA algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the dimensions ℓC and wC of the container.

Output: A feasible packing of the items in $\mathcal{P}$ into the container.

```
generate a population of intial solutions of size npop;
for g ← 1 to ngen do
    transfer the best nrep solutions from generation g − 1 to generation g;
    while the size of generation g is less than npop do
        if randomly chosen operator is crossover then
            select two parent solutions in generation g − 1;
            apply crossover to the selected parents to generate new offspring solutions for
            inclusion in generation g;
        else
            select parent solution in generation g − 1;
            apply standard mutation to the selected parent to generate new offspring
            solutions for inclusion in generation g;
    for i ← 1 to nmerge do
        select parent solution in generation g − 1;
        apply merger mutation to the selected parent to generate a new offspring os;
        if os is better than the worst solution ws in generation g then
            replace ws by os;
```

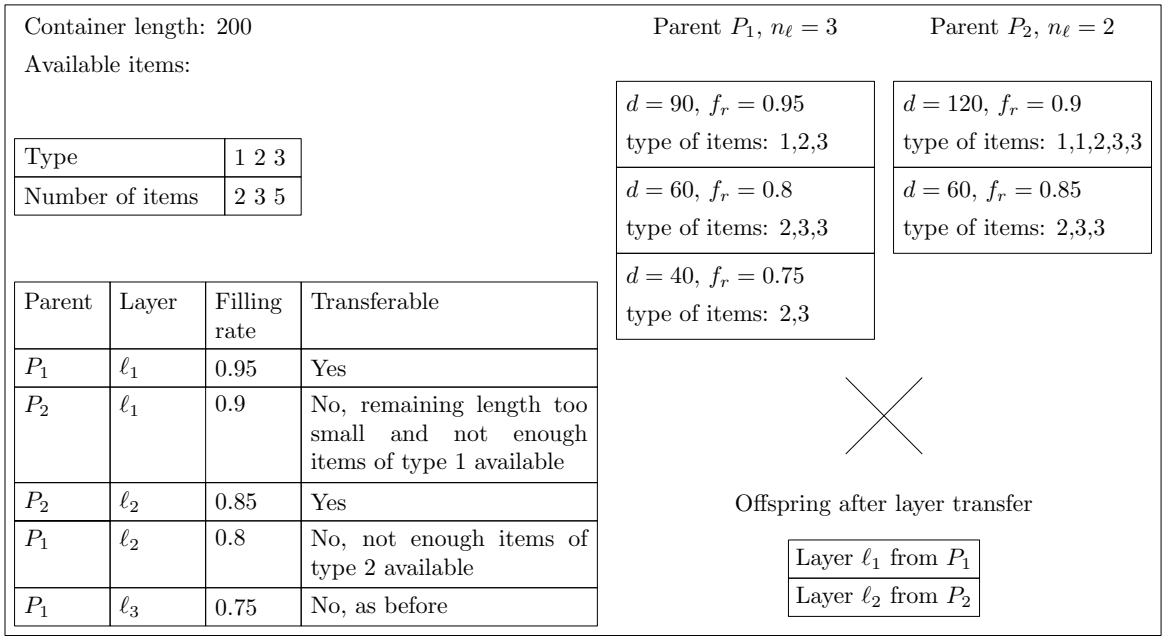


FIGURE 4.5: *Layer transfer within a crossover during the CLP-GA search of Bortfeldt and Gehring [26].*

If a standard mutation is carried out, a number $n\ell m$ of layers to be transferred is chosen at random from the interval $[1, \ldots, \frac{n\ell_P}{2}]$, where $n\ell_P$ is the total number of layers presented in the parent solution. The best $n\ell m$ layers of the parent are then transferred to the offspring solution. Before the next iteration begins, a merger mutation is applied to *nmerge* of the parents. The merger mutation transfers all but two randomly selected layers of the parent to the offspring solution. If the solution generated *via* the merger mutation is better than the worst solution of the incumbent generation, then the latter solution is discarded and is replaced by the offspring solution. The procedure terminates when a predefined total number *ngen* of generations have been generated.

An offspring solution, obtained through application of a crossover or mutation operator, might be incomplete. This partial solution may be completed by filling its remaining space with new layers. The procedure for completing an incomplete solution s_{in} is as follows. A set $Lv1$ of feasible layer variants for the first new layer of s_{in} is first determined. A layer variant consists of an ℓdp and its orientation. A layer variant is feasible if the ℓdp is free and it can still be placed in the container, respecting its orientation. For each variant in $Lv1$, a layer is generated and s_{in} is extended alternatively by each of these layers, resulting in a set $S1$ of temporary solutions.

Each of the temporary solutions $s \in S1$ is then extended separately by additional layers. A set Lvn of feasible layer variants with respect to the present state of s is determined accordingly. For each variant in Lvn, a layer is generated and the layer with the largest filling rate is added to s. The process is repeated until no further temporary solutions are extendable. The desired complete and best solutions in terms of the area covered are finally returned.

In order to reduce the time required to complete a solution, Bortfeldt and Gehring [26] imposed a restriction on permitted layer variants for $Lv1$ and Lvn. During the course of crossover, only the first $q\ell dp1\%$ and $q\ell dp2\%$ of all possible variants for $Lv1$ and Lvn are permitted, respectively. This means, for example, if 20 feasible layer variants exist for the first new layer and $q\ell dp1$ is set to 50%, that only the first 10 layer variants with the largest ℓdp area are considered for $Lv1$. In the case of mutation, only one variant is permitted, which is selected randomly from among the first $q\ell dp3\%$ of all possible variants for $Lv1$ and Lvn.

Given a feasible layer variant with fixed ℓdp, a complete layer is generated by means of a procedure called FillingLayer(). This procedure takes as input the selected ℓdp and the set $\mathcal{P}$ of unpacked items that is updated each time an item has been placed. The placings within a layer are collected in a layer record L which is finally returned. A pseudocode description of the procedure FillingLayer() may be found in Algorithm 4.4 and the working of the procedure is as follows.

Algorithm 4.4: **Fillinglayer()**

```
Input  : A layer defining piece ℓdp, and a list of unpacked items P.
Output: A layer record L.
L ← ∅;
initialise space stack Sstack, insert the layer h(ℓdp) × W as a single residual space;
remove ℓdp from P;
while Sstack ≠ ∅ do
    set the current residual space Scurr equal to the uppermost element in Sstack and
    remove this element from Sstack;
    if Scurr includes a placing then
        generate two daughter spaces within Scurr in accordance with the present placing;
        add the daughter spaces to Sstack;
        add the placing in Scurr to the layer record L;
    else
        determine the pair of items, pc and ps, with maximum total area that fits in Scurr;
        if one item is thus found then
            place the item in the reference corner of Scurr;
            generate two daughter spaces within Scurr;
        else
            place pc in the reference corner of Scurr;
            place ps in front of or beside pc;
            generate two daughter spaces within Scurr, taking into account the relative
            position of ps;
        add the placings in Scurr to L;
        remove the corresponding items from P;
        insert the daughter spaces into Sstack;
```

The layer record is initialised as empty. The full layer rectangle acts as the first residual space and is filled by the ℓdp in its reference corner. It is then inserted into a space stack Sstack containing residual spaces. The procedure next executes the following loop. At each iteration, the uppermost residual space of Sstack is removed and processed as the current residual space Scurr. If Scurr already includes a placing (as in the case of the first residual space, for example), then two new residual or daughter spaces are generated and inserted as empty spaces in Sstack. If Scurr is still empty but cannot accommodate any unpacked items, it is discarded. Otherwise the pair of items with the maximum total area that can be placed completely within Scurr is determined. If only one item is thus found, it is packed in the reference corner of Scurr and the respective placing is inserted in L. The daughter spaces are generated and inserted as empty spaces in Sstack. If, however, two items are found to fit into Scurr, one item (pc) is placed in the reference corner of Scurr and this placing is added to L. The other item (ps) is packed in front of or beside pc. The daughter spaces are generated and ps is placed in the reference corner of one of these spaces. The procedure terminates when no residual spaces remain.

The SPGAL for the Strip Packing Problem

The overall procedure of the SPGAL is provided in pseudocode form in Algorithm 4.5 and is outlined as follows. Initially a starting solution *Sstart* is calculated by means of the BFDH* heuristic, as described in §3.1. The resulting packing height initialises the minimum container length $\ell Cbest$ and the best solution *Sbest* is set to *Sstart*. The next step depends on the instance size. If the number of items in the instance exceeds the limit *nplarge*, only a subset $\mathcal{P}'$ of $\mathcal{P}$ is used to calculate further solutions. The subset $\mathcal{P}'$ is obtained by first sorting the layers in *Sstart* in ascending order according to the filling rate, then successively adding all items placed in each layer until a predefined limit *npsmall* of items is reached. The remaining layers of *Sstart* with the largest filling rates are kept in the partial solution *Skept*.

At this stage, the current container length ℓC is initialised as the difference $\ell Cbest - 1 - \ell(Skept)$, where $\ell(Skept)$ is the length of the partial solution *Skept*. The following loop is then executed until no solution including all items in $\mathcal{P}'$ is found. A container loading problem instance with the container dimensions ℓC, wC and the set $\mathcal{P}'$ is solved by means of the CLP-GA. The union of the new (partial) solution and the kept partial solution *Skept* represents a best solution to the corresponding SPP instance. Its height is equal to $\ell(s) + \ell(Skept)$. The container length is reduced by at least one unit before the next iteration begins.

Algorithm 4.5: **SPGAL algorithm**

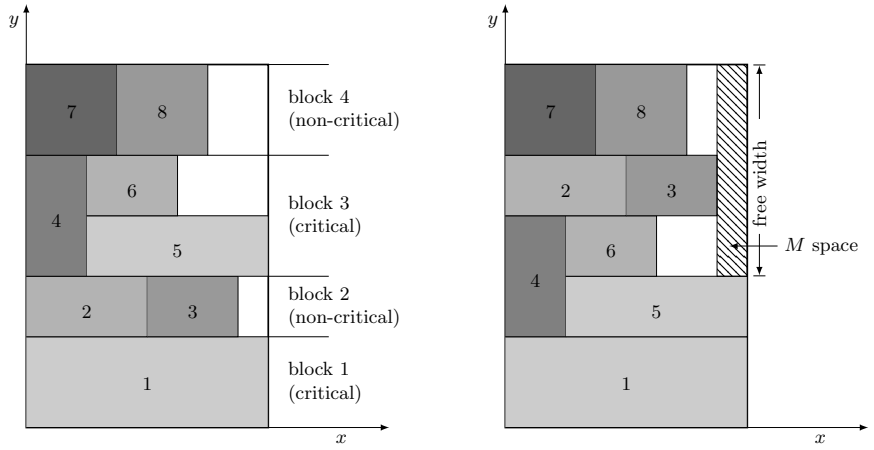
Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.
Output: A feasible packing of the items in $\mathcal{P}$ into a strip of width W.

```
calculate a starting solution Sstart by means of the heuristic BFDH*;
Sbest ← Sstart, ℓCbest ← ℓ(Sstart);
if n > nplarge then
    determine a suitable subset P' and keep the remaining layer of Sstart with the largest
    filling rates in Skept;
else
    P' ← P, Skept ← ∅;
initialise a container length ℓC ← ℓCbest − 1 − ℓ(Skept);
while the obtained solution s includes all items in P' do
    apply CLP-GA for a current length ℓC to the list of items P' and obtain a new
    solution s;
    Sbest ← s ∪ Skept, ℓCbest ← ℓ(s) + ℓ(Skept);
    ℓC ← ℓ(s) − 1;
```

The packing output produced by the SPGAL algorithm satisfies the guillotine constraint due to the layer structure of solutions. Since this required layer structure prohibits items from protruding over layer borders, space may remain available between two successive layers which could be filled. Bortfeldt [25] thus proposed a post-optimisation step after execution of the SPGAL algorithm in an attempt to fill such gaps so as to improve the solution quality. This heuristic consists of moving defined items in a layer into an adjacent layer and to use the available space thus created.

The post-optimisation heuristic takes a full packing layout as input and its working comprises three phases. In the *analysis phase* a block structure is determined for each layer of the packing layout (as illustrated in Figure 4.6(a)). Each layer is typically split into several blocks with one or more items arranged in each block. The length of a block is taken as the sum of the

x-dimensions of all items in the block, while its width is defined as the y-dimension of the lowest item in the block. These blocks are then divided into critical and non-critical blocks. The length (x-dimension) of a critical block is equal to the layer length.



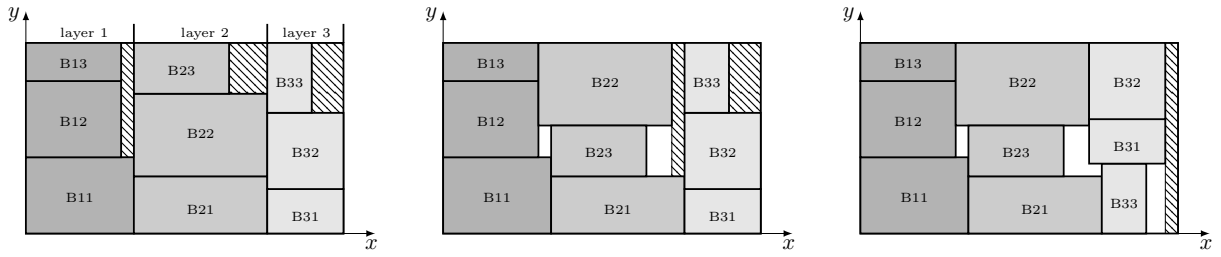
(a) *The block structure of a layer* (b) *The same layer after reorganisation*

FIGURE 4.6: *The block structure and reorganisation of a layer.*

During the *reorganisation phase*, a rectangular free space, called M space, is created for every single layer (as illustrated in Figure 4.6(b)). The width dimension of this space, called the free width, is maximised by reorganising its blocks. This is achieved by rearranging the blocks in the layer in such a way that first the critical and then the non-critical blocks follow one another without any gaps. During the *displacement phase*, a limited number of layers of maximum free width are selected. All possible permutations of these layers are subjected to a displacement process; the best packing layout (with the shortest height over all permutations) is completed by including the remaining layers and is finally returned.

The displacement process for a particular permutation of the selected layers is illustrated in Figure 4.7. The selected layers, together with their rearranged blocks, are shown in Figure 4.7(a). Starting with the second layer of the permutation, a subset of the critical blocks in layer $i > 1$ which achieves a maximum sum of widths, but is smaller than the free width of the layer $i - 1$, is determined. These critical blocks are placed into the M space of layer $i - 1$ in the left-most position and the remaining blocks in the layer i are rearranged, while the corresponding M space is updated. The displacement process for the second layer is shown in Figure 4.7. Block B22 is the only critical block that fits into the M space of the first layer, it is thus exchanged with B23 and pushed to the left into the M space of layer 1. Blocks B23 and B21 are placed next to each other, resulting in a new M space for layer 2, which is the area above B22 and B23. Following the same process, critical blocks B31 and B32 of layer 3 are displaced and pushed into the M space of layer 2. This results in a new M space for layer 3 with width equal to the container width (as illustrated in Figure 4.7(c)), indicating a reduction of the strip height.

Example 8 *An example of output produced by the SPGAL algorithm, when applied to the instance $\mathcal{I}$ of Figure 3.1, is shown in Figure 4.8. The parameter settings of the algorithm were as follows. The population size npop was taken as 50, the number of solutions to be reproduced nrep and to be generated through merger mutation nmerge per generation were both set to 10. The crossover probability was selected as 67% and the mutation probability as 33%. The percentages $q\ell dp1$, $q\ell dp2$ and $q\ell dp3$ depend on the number nptypes of item types of the given instance; their values are summarised in Table 4.1. The number ngen of generations per instance also depends*



(a) *Three layers with rearranged blocks Bij (i, j=1,2,3)*

(b) *Same layers after displacement of the second layer*

(c) *Same layer after displacement of the third layer*

FIGURE 4.7: *Results of the displacement process for a particular permutation of the selected layers. Dashed areas indicate the rectangular free space called the M space.*

on the number of items n in the instance. Its value was taken as follows: ngen was equal to 1 000 if $n \in [1, 60]$*, equal to 500 if* $n \in [61, 100]$*, or 100 otherwise. The parameters nplarge and npsmall were taken as 199 and 100, respectively.* ■

nptypes in	[1,40]	[41,60]	[61,200]	> 200
$q\ell dp1$	100	66	10	5
$q\ell dp2$	100	66	10	5
$q\ell dp3$	33	33	33	33

TABLE 4.1: *Values of the percentages* $q\ell dpi$ *for* $i = 1, 2, 3$ *in the SPGAL algorithm.*

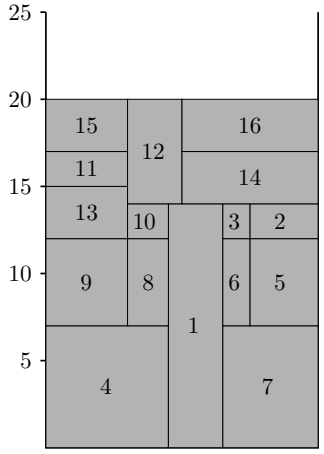


FIGURE 4.8: *An example of a packing solution obtained when applying the SPGAL algorithm to the instance* $\mathcal{I}$ *in Figure 3.1. The resulting packing height is* $\mathtt{H} = 20$*.*

4.2.4 The Reactive GRASP Algorithm

The greedy randomised adaptive search procedure, or GRASP algorithm, was first introduced by Feo and Resende [63] and is an iterative process consisting of two phases: A construction phase and an improvement phase. During the construction phase, a solution is constructed step-by-step by adding one new element at a time to a partial solution. The new element is drawn at random from a limited list of candidates, which contains the best elements according to a certain criterion. A local search is then performed during the improvement phase in order to

improve the solution generated during the construction. This is achieved by substituting some elements of the solution or applying heuristic techniques.

During the construction phase of the reactive GRASP algorithm proposed by Alvarez-Valdés *et al.* [5] for the SPP, the algorithm finds the lowest and the left-most available location in the strip or the constructed partial solution, and records in a list L all possible blocks of items that fit into it. A location is defined by a skyline segment represented by a pair (ω, ℓ), where ω is the width and ℓ is the level of the segment. A block of items is obtained by placing items of the same type next to one another (disallowing rotation) such that its total width is at most the width of the selected location. If the width of the lowest available location found cannot accommodate any unpacked items, then it is raised to the height of its shortest neighbour. Each block b in L is associated with a score s_b, which serves as an attribute for choosing the best block to fill the chosen location. Once this step has been performed, the algorithm applies a random selection procedure by choosing a block b_p at random from a restricted set C of candidates based on the scoring strategy.

In the next step, before packing b_p, the algorithm first tests whether it is the highest block in L. If this is the case, it is placed in the chosen location. Otherwise, the algorithm calculates the empty area E defined by the height if the tallest unpacked block b_t were to be placed in the chosen location, comparing it with the area M of remaining unpacked items together with an estimation of the unavoidable waste involved in the process $U = (W \times LB - A)/4$, where LB is a lower bound on the required height and A is the total area of the pieces. If $E > \lambda(M + U)$, where λ is a parameter chosen randomly in the real interval (0.9, 1.6), then b_t is selected for packing, otherwise b_p is packed in the chosen location. A block is placed either in the left-most position or adjacent to the tallest neighbour or adjacent to the shortest neighbour. Once a block has been packed, the skyline profile is updated and the entire process is repeated until a complete solution is obtained.

Alvarez-Valdés *et al.* [5] conducted several preliminary computational experiments to evaluate different strategies in order to choose the best ones for inclusion in their GRASP algorithm. Their first study consisted of comparing four different scoring criteria for choosing the best block of items to pack during the construction phase. The first criterion involves the width, breaking ties by sorting items according to non-increasing height. The second scoring criterion involves the block width together with a relative weight of its height, $w + kh$, where $k \in (0.01, 0.75)$. The third criterion is similar to the second, but involves a fixed value of k, namely 0.5. The last criterion is called the *best profile criterion*, and consists of choosing the block whose height is the most similar to the difference between the level height of the chosen location and the level height of its neighbour. According to their computational study results, none of these rules generated the best results consistently. The second criterion, $w + kh$, where k is taken randomly from the interval (0.01, 0.75), is employed in this dissertation.

Alvarez-Valdés *et al.* [5] also considered and evaluated four different selection criteria based on four distinct restricted sets of candidate items. The first set is $C = \{j \mid s_j \geq s_{\min} + \gamma(s_{\max} - s_{\min})\}$, where $s_{\max} = \max\{s_b \mid b \in L\}$, $s_{\min} = \min\{s_b \mid b \in L\}$, and γ is a parameter to be determined $(0 < \gamma < 1)$. The second set is $C' = \{j \mid s_j \geq \gamma(s_{\max})\}$. The third criterion is to choose a block randomly from among the best $100(1 - \gamma)\%$ of all blocks in L. The last criterion is to select a block from L with a probability proportional to its score $p_b = s_b / \sum s_b$. Their computational results, for a value of $\gamma = 0.5$, indicated that the first random selection criterion yields good results and so this criterion is employed in this dissertation. Furthermore, they considered several strategies of changing the value of γ as a function of search iterations. The first and second strategies consist of choosing the value of γ randomly from the intervals $[0.4, 0.9]$ and $[0.25, 0.75]$ at each iteration, respectively. According to the third strategy, γ takes

one of the following values: 0.5, 0.6, 0.7, 0.8, or 0.9. In the fourth strategy the value of γ is fixed at 0.75. The last strategy is called the reactive GRASP and involves choosing γ randomly from a set of discrete values. Initially all values have the same probability of being chosen. After a certain number of iterations the probabilities are, however, modified: Those corresponding to values of γ which have produced good solutions are increased while the remaining probabilities are decreased. This last strategy has been demonstrated to provide good results and is therefore employed in this dissertation.

Four different improvement methods have also been considered by Alvarez-Valdés *et al.* [5] aimed at improving the solution formed during the construction phase. According to the first method, the last 20% items packed during the construction phase are removed and a closed bin of width equal to the width of the strip and height equal to the height of the solution reduced by one unit is considered, containing the remaining items. The constructive algorithm developed by Alvarez-Valdés *et al.* [4] for the non-guillotine cutting stock problem is applied to pack the items thus removed into the bin. If all items can be packed into the bin, an improved solution to the original SPP problem instance is obtained. The task of the second method is to remove the item that defines the maximum height of the initial solution and to place it in some waste location in a lower level of the strip. If the item exceeds the dimension of the waste space, the items overlapping it are removed. The items thus removed are thereafter packed using the constructive procedure. The third improvement method consists of eliminating the last k% items of the initial solution and filling the empty space by invoking the construction algorithm. If no improvement in terms of the solution height is obtained for the chosen percentage, further items are removed until the new height is smaller than the original packing height. The fourth method is similar to the third, except that in this case all items with upper edges exceeding a height of βH are removed, where H is the height of the initial solution and $0 < \beta < 1$. The comparative study results of Alvarez-Valdés *et al.* indicated that the third and fourth methods provided identical results. The third method is therefore employed in this dissertation. A pseudocode representation of the reactive GRASP is presented in Algorithm 4.6.

Example 9 *An example of a solution returned by the GRASP algorithm, when applied to the instance $\mathcal{I}$ of Figure 3.1, is given in Figure 4.9. The configuration of the algorithm is based on the best strategies described above, and the value of ζ was fixed at 10, as suggested in [5].* ■

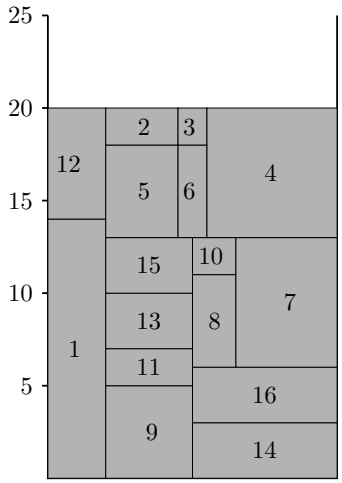


FIGURE 4.9: *An example of a packing solution obtained when applying the reactive GRASP algorithm to the instance $\mathcal{I}$ of Figure 3.1. The resulting packing height is* $\mathtt{H} = 20$.

Algorithm 4.6: **Reactive GRASP algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.

Output: A feasible packing of the items in $\mathcal{P}$ into a strip of width W.

$\mathcal{D} \leftarrow \{0.1, 0.2, \ldots, 0.9\}$, set of possible values for γ;
Pool$\gamma \leftarrow \emptyset$, a set of solutions obtained the value $\gamma \in \mathcal{D}$;
$p_\gamma \leftarrow 1/|\mathcal{D}|$, an initial probability of choosing a particular value $\gamma \in \mathcal{D}$;
$H_{\text{best}} \leftarrow \infty$, the initial best packing height solution;
$H_{\text{worst}} \leftarrow 0$, the initial worst packing height solution;
NumIter $\leftarrow 0$;
while *NumIter* $<$ *MaxIter* **do**
    choose $\gamma*$ from $\mathcal{D}$ with probability $p_{\gamma*}$;
    apply the construction phase with $\gamma*$, obtaining a solution S of height H;
    apply improvement phase to S, obtaining a new solution S' of height H';
    if $H' < H_{best}$ **then**
        $H_{best} = H'$;
    if $H' > H_{worst}$ **then**
        $H_{worst} = H'$;
    Pool$\gamma* \leftarrow$ Pool$\gamma* \cup \{S'\}$;
    if *mod(NumIter,200)=0* **then**
        $eval_\gamma \leftarrow \left(\frac{\text{mean(Pool}\gamma) - S_{\text{worst}}}{S_{\text{best}} - S_{\text{worst}}}\right)^\zeta$, for each value $\gamma \in \mathcal{D}$;
        $p_\gamma \leftarrow \frac{eval_\gamma}{\sum_{\gamma' \in \mathcal{D}} eval_{\gamma'}}$, for each value $\gamma \in \mathcal{D}$;
    NumIter $\leftarrow$ *NumIter* $+ 1$;

4.2.5 The Two-stage Intelligent Search Algorithm

The two-stage ISA proposed by Leung *et al.* [116] combines a local search algorithm and an SA algorithm in an attempt to solve the 2D SPP. The local search algorithm first sorts the list of items to be packed according to non-increasing perimeter size, stores the ordered list in a variable *bestX*, and then calls a constructive heuristic to generate a complete solution for which the resulting packing height is stored in a variable *besth*. Thereafter, it executes a loop which consists of swapping pairs of items in the current list so as to obtain new ordered lists, after which the packing height of these new lists are computed by means of the constructive heuristic. If the value of a new packing height is less than the current height *besth*, it becomes the new *besth* and the new ordered list replaces the current list *bestX*. Otherwise the current values are kept. The loop terminates when all pairs of items have been swapped according to some predetermined order.

The list *bestX* obtained upon execution of the local search algorithm is further used as initial packing order in the SA algorithm. Each iteration of the inner loop of the SA algorithm consists of swapping the order of two randomly selected items in *bestX*, and then calling the constructive heuristic (the same heuristic as that employed in the local search algorithm) to compute the respective packing height. An improvement of the packing height is always accepted, while a non-improvement in the packing height is accepted according to a given probability. In order to enhance the search capability of the SA algorithm, Leung *et al.* designed a multi-start strategy encouraging the process to examine unvisited regions in solution space. At the end of each

iteration, the newly generated list of items is sorted, either according to non-increasing width or non-increasing perimeter. The loop terminates when a predefined stopping criterion is satisfied. The constructive heuristic HeuristicPacking() described in §3.5 was adopted by Leung *et al.* as packing heuristic in their ISA algorithm. A pseudocode representation of the ISA algorithm is presented in Algorithm 4.7. The local search and SA algorithms embedded in the ISA procedure may be found in Algorithms 4.8 and 4.9, respectively.

Algorithm 4.7: **Two-stage Intelligent Search Algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.
Output: The best packing height *besth* and the best ordering list *bestX*.

```
1 LocalSearch();
2 SimulatedAnnealing();
3 return besth and bestX;
```

Algorithm 4.8: **Local Search algorithm** (LocalSearch())

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.
Output: The best packing height *besth* and the best ordering list *bestX*.

```
1  sort all unpacked items according to non-increasing perimeter size to obtain a packing
   order X;
2  besth ← HeuristicPacking(X), bestX ← X;
3  for i ← 1 to n − 1 do
4      for j ← i + 1 to n do
5          swap the order of items i and j in X and obtain a new ordering X′;
6          currenth ← HeuristicPacking(X′);
7          if currenth < besth then
8              besth ← currenth;
9              bestX ← X′, X ← X′;
10 return besth and bestX;
```

Example 10 *An example of a solution returned by the ISA algorithm, when applied to the instance $\mathcal{I}$ of Figure 3.1, is given in Figure 4.10. The cooling schedule employed in the SA algorithm was the geometric schedule with cooling rate 0.93. The initial value of the temperature was taken as 0.5. The temperature was held constant for 16 moves (the total number of items in the instance). The stopping criterion was to complete 1 000 iterations.* ■

4.2.6 The Simple Randomised Algorithm

In 2013, Yang *et al.* [162] proposed an improved ISA, which they referred to as the *simple randomised algorithm* (SRA). There are two major differences between the ISA and the SRA. The first is related to the scoring rule employed in the constructive heuristic embedded within each search procedure: Five cases are implemented in the ISA, while eight distinct cases are considered in the SRA. The second difference occurs during the second phase of the search procedures: The ISA applies an SA algorithm in an attempt to improve solutions, while a simple randomisation without any parameter settings is used instead in the SRA.

Algorithm 4.9: **Simulated Annealing algorithm** (SimulatedAnnealing())

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, the strip width W, an initial temperature T_0, and a cooling rate α.
Output: The best packing height *besth* and the best ordering list *bestX*.

```
1  generate an initial solution X using LocalSearch();
2  T ← T_0;
3  while the stopping criterion is not yet satisfied do
4      for i ← 1 to L do
5          randomly select two items j and k in X;
6          obtain a new ordering X' by swapping the order of j and k;
7          currenth ← HeuristicPacking(X');
8          if currenth < besth then
9              besth ← currenth;
10             bestX ← X', X ← X';
11         else
12             if exp((besth − currenth)/T) ≥ (rand(0, 1)) then
13                 X ← X';
14     T ← αT;
15 return besth and bestX;
```

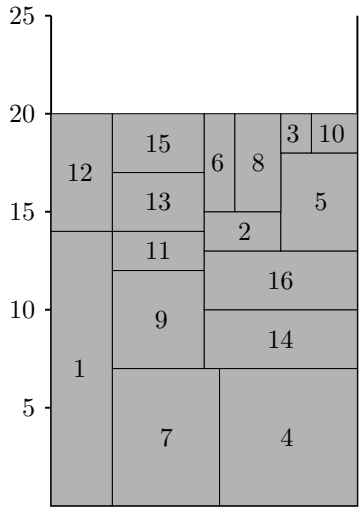
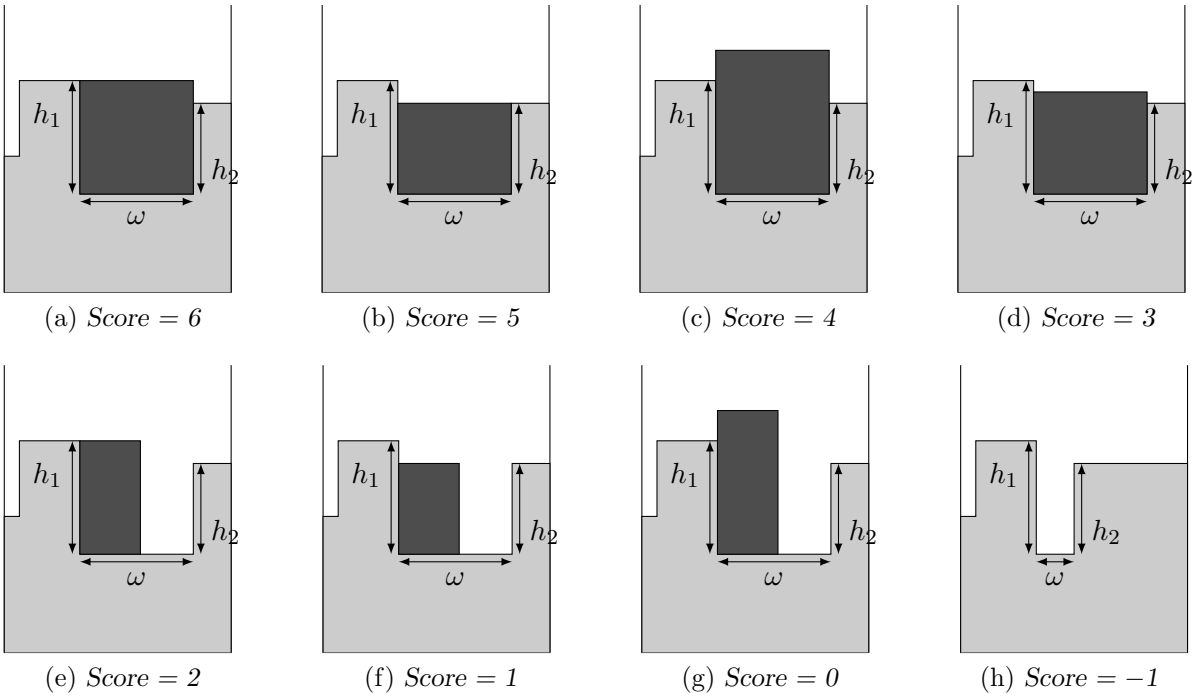


FIGURE 4.10: *An example of a packing solution obtained when applying the ISA algorithm to the list of items $\mathcal{I}$ in Figure 3.1. The resulting packing height is* $\mathtt{H} = 20$.

The SRA first applies a local search algorithm in order to obtain a good packing solution. The local search algorithm here is similar to LocalSearch() described in Algorithm 4.8, except that a non-increasing ordering of items according to height is employed as a sorting strategy during the first step instead. The packing order obtained upon execution of this process is used as initial packing order for the randomised algorithm. The inner loop of the randomised algorithm follows the same steps as in the algorithm SimulatedAnnealing() of the ISA, described in Algorithm 4.9. The probability of accepting a non-improving solution is, however, different. In the SA algorithm, this probability depends on the temperature parameter and the value of the current and the best-found solutions, while in the randomised algorithm no parameter

is required, the probability depends only on the current and the best solutions found. The procedure is executed until a predefined stopping condition is satisfied.

The constructive heuristic HeuristicPacking() described in §3.5 was again adopted by Yang *et al.* as heuristic packing algorithm in their SRA algorithm. Using the same notation as in §3.5, the new scoring rule utilised by Yang *et al.* in their constructive heuristic is summarised in Table 4.2 for the case $h_1 \geq h_2$ (there are eight similar cases for $h_1 < h_2$) and is illustrated by an example in Figure 4.11. A pseudocode listing of the SRA algorithm is provided in Algorithm 4.10.



(a) *Score* = 6 (b) *Score* = 5 (c) *Score* = 4 (d) *Score* = 3

(e) *Score* = 2 (f) *Score* = 1 (g) *Score* = 0 (h) *Score* = −1

FIGURE 4.11: *An example of the scoring rule used by Yang* et al. *[162]. The parameters* h_1, h_2, *and* ω *represent the height of the left wall, the height of the right wall, and the width of a selected available space, respectively. The light-shaded area represents items already packed, while the dark-shaded area represents the newly packed item being scored.*

If	Conditions	Score
	$\omega =$ item.width and $h_1 =$ item.height	6
	$\omega =$ item.width and $h_2 =$ item.height	5
	$\omega =$ item.width and $h_1 <$ item.height	4
$h_1 \geq h_2$	$\omega =$ item.width and $h_1 >$ item.height	3
	$\omega >$ item.width and $h_1 =$ item.height	2
	$\omega >$ item.width and $h_2 =$ item.height	1
	$\omega >$ item.width and $h_1 \neq$ item.height	0
	$\omega <$ item.width	−1

TABLE 4.2: *The scoring rule utilised by Yang* et al. *[162] to determine the best item that fits into a selected available space* s. *The parameters* h_1, h_2, ω *are the height of the left wall, the height of the right wall and the width of* s, *respectively.*

Example 11 *An example of a solution returned by the SRA algorithm, when applied to the instance* $\mathcal{I}$ *of Figure 3.1, is given in Figure 4.12. The stopping criterion adopted was the execution of 1 000 search iterations.* ■

Algorithm 4.10: **Simple Randomised Algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.

Output: The best packing height *besth* and the best packing order *bestX*.

```
1  X ← LocalSearch();
2  while the stopping criterion is not yet satisfied do
3      for i ← 1 to n do
4          randomly select two items j and k in X;
5          obtain a new packing order X′ by swapping the order of items j and k;
6          currenth ← HeuristicPacking(X′);
7          if currenth < besth then
8              besth ← currenth;
9              bestX ← X′, X ← X′;
10         else
11             p ← currenth / (currenth + besth);
12             if p < (rand(0,1)) then
13                 X ← X′;
14     randomly select a sorting rule, according to non-increasing perimeter, area or width;
15 return besth;
```

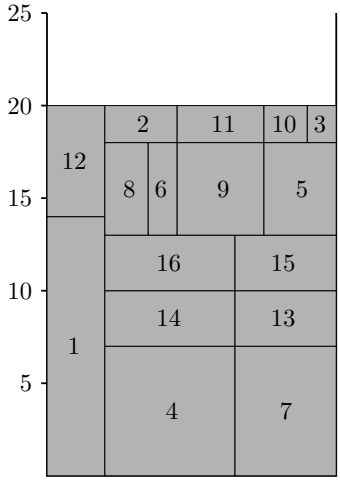


FIGURE 4.12: *An example of a packing solution obtained when applying the SRA algorithm to the list of items $\mathcal{I}$ in Figure 3.1. The resulting packing height is* $\mathtt{H} = 20$.

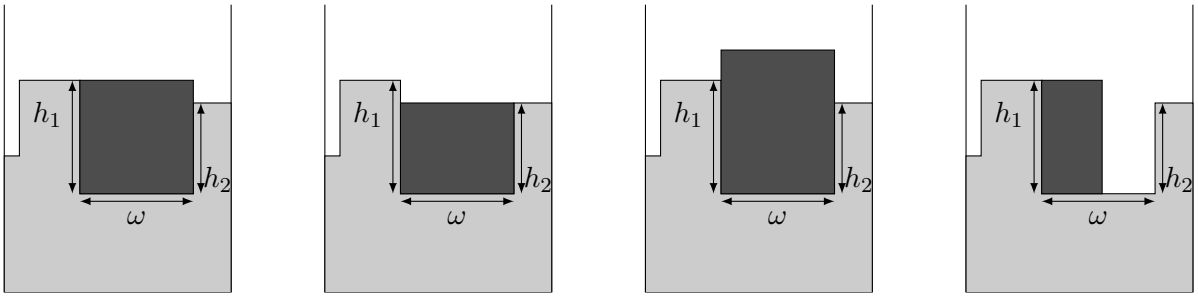
4.2.7 The Improved Algorithm

In 2016, Wei *et al.* [158] proposed an efficient intelligent search algorithm, referred to as the *improved algorithm* (IA), which is based on the ISA of Leung *et al.* [116]. It involves three stages: greedy selection, local improvement, and randomised improvement. For a given list of items to be packed, the greedy selection algorithm first generates four different packing orders, which are obtained by sorting the list of items according to non-increasing perimeter size, area, height, and width, respectively. The packing layout, as well as the packing height, of each of these lists is then evaluated by means of a constructive heuristic. The constructive heuristic employed for this purpose is HeuristicPacking(), employed by Yang *et al.* [162] in their SRA

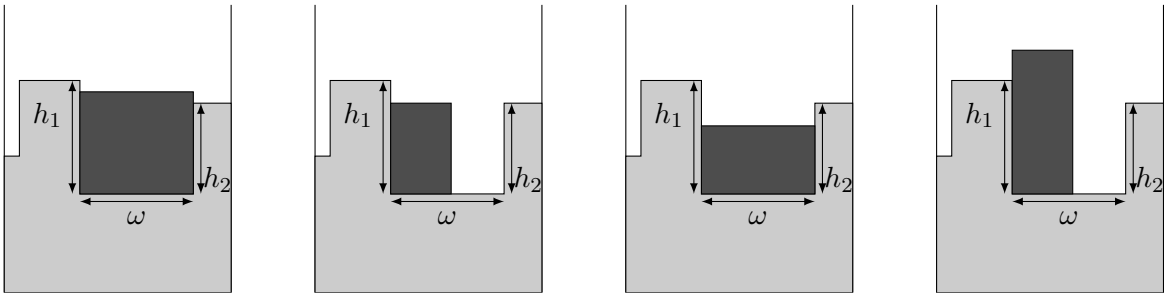
algorithm. The scoring rule utilised by Wei *et al.* in their constructive heuristic is, however, different from that employed by Yang *et al.* [162]. Using the same notation as in §4.2.6, this new scoring rule is summarised in Table 4.3 and is illustrated by an example in Figure 4.13.

If	Conditions	Score
	$\omega =$ item.width and $h_1 =$ item.height	7
	$\omega =$ item.width and $h_2 =$ item.height	6
	$\omega =$ item.width and $h_1 <$ item.height	5
$h_1 \geq h_2$	$\omega >$ item.width and $h_1 =$ item.height	4
	$\omega =$ item.width and $h_1 >$ item.height	3
	$\omega >$ item.width and $h_2 =$ item.height	2
	$\omega =$ item.width and $h_2 >$ item.height	1
	$\omega >$ item.width and $h_1 <$ item.height	0

TABLE 4.3: *The scoring rule employed in the IA algorithm of Wei* et al. *[158] for determining the best-suited item to pack into a selected available space. The parameters h_1, h_2, and ω denote the height of the left wall, the height of the right wall and the width of the available space, respectively.*



(a) *Score = 7* (b) *Score = 6* (c) *Score = 5* (d) *Score = 4*



(e) *Score = 3* (f) *Score = 2* (g) *Score = 1* (h) *Score = 0*

FIGURE 4.13: *Examples of the scoring rule utilised by Wei* et al. *[158]. The parameters h_1, h_2, and ω are the height of the left wall, the height of the right wall, and the width of a selected available space, respectively. The light-shaded area represents items already packed, while the dark-shaded area represents the newly packed item being scored.*

At the end of the greedy selection procedure, two packing orders that yield the smallest packing heights are selected and saved. At this stage, the IA algorithm executes a local search algorithm twice, taking the aforementioned two packing orders produced during the greedy selection stage as the initial solutions. The detailed procedure followed during the local search is similar to that in LocalSearch(), described in Algorithm 4.8. The only difference is that the first step of LocalSearch() is omitted in this case. The two new packing orders thus obtained are further used as initial packing orders during the randomised improvement stage. The objective of this latter stage is to further explore the search space with a view to improve the quality of the solutions obtained so far. The procedure RandomisedImprovement() involves randomly swapping

the positions of two items in the current ordering. If the new solution is better than the previous one, it is accepted. The procedure terminates when a pre-defined stopping criterion is satisfied.

A pseudocode description of the IA algorithm is presented in Algorithm 4.11. The greedy selection and randomised improvement algorithms embedded in the IA procedure are similarly described in Algorithms 4.12 and 4.13, respectively.

Algorithm 4.11: Improved Algorithm

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.
Output: A packing height *besth* and the best packing order *bestX*.

```
GreedySelection();
LocalSearch();
RandomisedImprovement();
return besth and bestX;
```

Algorithm 4.12: **Greedy Selection algorithm** (GreedySelection())

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.
Output: Two packing orders with the smallest packing heights.

```
Sort all items by perimeter to obtain X_p and H_p ← HeuristicPacking(X_p);
Sort all items by area to obtain X_a, H_a ← HeuristicPacking(X_a);
Sort all items by height to obtain X_h, H_h ← HeuristicPacking(X_h);
Sort all items by width to obtain X_w, H_w ← HeuristicPacking(X_w);
besth ← min{H_p, H_a, H_h, H_w} and save the sequence X_B that leads to besth;
save the sequence X_b that leads to the second smallest height;
```

Example 12 *An example of a solution returned by the IA algorithm, when applied to the instance $\mathcal{I}$ of Figure 3.1, is given in Figure 4.14. The stopping criterion was specified as having carried out 1 000 search iterations.* ■

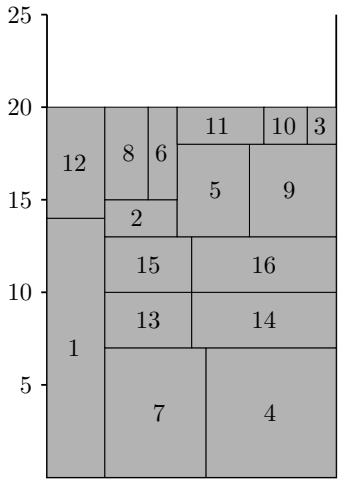


FIGURE 4.14: *An example of a packing solution obtained when applying the IA algorithm to the list of items $\mathcal{I}$ in Figure 3.1. The resulting packing height is* H = 20.

Algorithm 4.13: **Randomised Improvement algorithm** (RandomisedImprovement())

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.
Output: The best packing height *besth*.

```
1  generate two initial packing solutions X_B, X_b by means of LocalSearch();
2  while the stopping criterion is not yet satisfied do
3      X ← randomly select one sequence from X_B and X_b;
4      CurrentH ← HeuristicPacking(X);
5      for i ← 1 to n do
6          randomly select two items j and k in X;
7          obtain a new ordering X' by swapping the order of j and k;
8          H ← HeuristicPacking(X');
9          if H < currenth then
10             currenth ← H;
11             X ← X';
12             update the besth if currenth is better;
13 return besth;
```

4.3 Chapter Summary

The first section of this chapter contained descriptions of the basic working of GAs and the method of SA in the context of packing problems. The various genetic operators and parameters involved in GA implementations were reviewed in §4.1.1. This was followed by a similar description of the working of the SA search technique in §4.1.2.

The second section of the chapter contained a review of various metaheuristic algorithms developed specifically for the 2D SPP. The concept of hybrid GAs was discussed first in §4.2.1, and this was followed by a similar discussion of hybrid SAs in §4.2.2. Thereafter, the SPGAL algorithm of Bortfeldt [25] was reviewed in §4.2.3. The current state-of-the-art strip packing algorithms were then detailed. These included the reactive GRASP algorithm of Alvarez-Valdés *et al.* [5] (in §4.2.4), the two-stage ISA proposed by Leung *et al.* [116] (in §4.2.5), the simple randomised algorithm of Yang *et al.* [162] (in §4.2.6), and the improved algorithm developed by Wei *et al.* [158] (in §4.2.7).

Part II

Clustering of Data

CHAPTER 5

Benchmark Instances

Contents

The relative performances of all algorithms considered throughout this dissertation are evaluated and compared in respect of a wide variety of problem instances from the strip packing literature. Two classes of benchmark instances are described in this chapter. The first class consists of zero-waste problem instances for which the respective optimal solutions are known and do not contain wasted regions (regions of the strip not occupied by items). This class of benchmark instances comprises nine data sets. The second class consists of non-zero-waste instances for which optimal solutions are not known in all cases and those with optimal solution known, but involving some wasted regions. This second class of problem instances comprises eleven data sets. The problem generators and methods employed to generate each of these benchmark instances are outlined in this chapter.

5.1 Zero-waste Problem Instances

The class of zero-waste SPP instances considered in this dissertation consists of nine data sets, as mentioned. These data sets are described in this section.

5.1.1 The Instances of Jakobs (J)

Jakobs [99] generated two SPP benchmark instances by taking a stock rectangle of height 15 and width 40, and randomly cutting it into smaller pieces. One problem instance comprises 25 rectangular items and the other instance 50 items. Optimal solutions to these problem instances are therefore known and are not necessarily guillotineable. Only Bortfeldt [25] has employed these instances to compare the performance of his packing algorithm with those of other algorithms in the context of the SPP. These benchmark instances were downloaded from [60].

5.1.2 The Instances of Hifi (SCP)

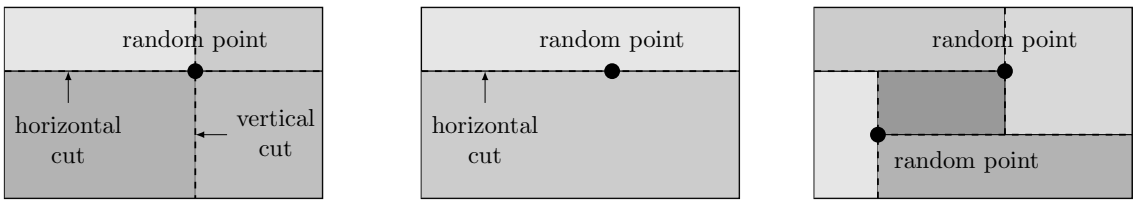
Hifi generated 25 problem instances in order to test the relative performances of his packing algorithms in 1998 [81]. He did not, however, provide details on the construction of these instances. The only available information concerning these problem instances is the fact that they were randomly generated. The sizes of these benchmark instances are relatively small and they have been ordered in order of increasing complexity. These data sets were obtained from [82] and optimal solutions are known for all these instances.

5.1.3 The Instances of Babu (babu)

Babu and Babu [8] generated a single SPP instance in order to evalualte the performance of his packing approach in 1999. He did not, however, give any details on the construction of this instance, except that the items are cut from an initial large sheet. The instance was downloaded from [117] and an optimal solution is known for the instance.

5.1.4 The Instances of Hopper and Turton (NT(n), NT(t) and C)

Details of the methods employed by Hopper and Turton to generate their benchmark problem instances may be found in [91, 92]. Three problem generators were developed to create both guillotineable and non-guillotineable problem instances with known optimal solutions. The first guillotineable problem generator consists of generating a large rectangle, selecting a random point in the rectangle, then partitioning the rectangle into four parts by means of a horizontal cut and a vertical cut through the point. A test problem is obtained by repeatedly executing this procedure until the number of rectangles created equals the required problem size. The second guillotineable problem generator follows the same steps as those in the first algorithm, except that only two rectangles are created per intersection point at each iteration. As in the two previous procedures, the non-guillotineable benchmark generator creates test problems by selecting a rectangle and splitting it into smaller ones. The working of this third algorithm is based on randomly selecting two points in the rectangle, then generating a pattern of five smaller rectangles in a non-guillotine manner. The splitting strategy embedded in each of the three problem generators is illustrated by means of an example in Figure 5.1(a)–(c).



(a) *Splitting an existing rectangle into 4 new rectangles by means of the first problem generator.*

(b) *Splitting an existing rectangle into 2 new rectangles by means of the second problem generator.*

(c) *Splitting an existing rectangle into 5 new rectangles by means of the third problem generator.*

FIGURE 5.1: *Rectangular item generation according to the three problem generators proposed by Hopper and Turton [91, 92].*

Using the aforementioned problem generators, Hopper and Turton generated three benchmark data sets, two of which are large sets (each containing 350 instances), while the other one is a smaller set (containing 210 instances). Half of the instances in the large benchmark sets [92] were generated by means of the first algorithm described above (the set labelled NT(t)), and the other half by means of the third algorithm described earlier (the set labelled NT(n)). Each of these two sets consists of seven categories, with each category comprising five instances containing an identical number of items. The dimensions of the rectangles in each instance are constrained by a maximum aspect ratio of 7. The smaller benchmark set [91] (the set labelled C or CP) also consists of seven categories, with each category containing three instances, each generated by a different problem generator. The optimal packing heights are known for each instance and these data sets were downloaded from [60].

5.1.5 The Instances of Burke, Kendall and Whitwell (N)

The benchmark problem instance generator proposed in 2004 by Burke *et al.* [29] takes as inputs the dimensions of an initial large rectangle, the number of items to be cut from this rectangle and a dimension constraint (the minimum dimension allowed) for any rectangle. The algorithm repeatedly makes random horizontal and vertical cuts to randomly selected rectangles — initially the specified large rectangle — such that the dimension constraint is satisfied, until the desired number of rectangles have been produced. The data set includes twelve instances, and were obtained from [153]. Optimal solutions to these benchmark instances are known and are guillotineable.

5.1.6 The Instances of Pinto and Oliveira (CX)

In 2005, Pinto and Oliveira [133] generated a set of benchmark problem instances in order to study the relative performances of their proposed SPP algorithms. They did not, however, give any details on the method of construction of these benchmark instances. The data set contains seven instances for which optimal solutions are known and which were downloaded from [117].

5.1.7 The Instances of Imahori and Yagiura (IY)

Imahori and Yagiura [94] provided no information about the method of generation of their 2010 SPP benchmark instances. The data sets consist of seventeen categories, with each category comprising ten instances containing an identical number of items. The number of items ranges from 16 to 1 048 576. These instances have been used by Leung *et al.* [116], Yang *et al.* [162],

and Wei *et al.* [158] to test the relative performances of their SPP algorithms, and were obtained from [95]. Optimal solutions are known for all 170 instances.

5.2 Non-zero-waste Instances

The class of non-zero-waste SPP problem instances considered in this dissertation comprises eleven data sets, which are described in this section.

5.2.1 The Instances of Christofides and Whitlock (cgcut)

The method used by Christofides and Whitlock to generate their benchmark problem instances in 1977 is described in detail in [40]. These benchmark instances were initially generated as a test case for the constrained two-dimensional cutting problem, and have previously been employed in the context of the SPP by Alvarez-Valdés *et al.* [5], Leung *et al.* [116], and Yang *et al.* [162].

The problem generator method starts with an initial rectangle R_0 of area $\mathcal{A}(R_0)$. Then m rectangles $R_1, \ldots, R_m$ are generated randomly by drawing the area of rectangle R_i $(i = 1, \ldots, m)$ from a uniform distribution on the range $[0, \mathcal{A}(R_0)/4]$. The height, $h(R_i)$, of R_i is an integer drawn from a uniform distribution in the range $[0, \mathcal{A}(R_i)]$ and its corresponding width is given by $\mathcal{A}(R_i)/h(R_i)$. The data set includes three instances with strip widths of 10, 70, 70, respectively, and an optimal solution is known for one of the instances. This data set was downloaded from [60].

5.2.2 The instances of Bengtsson (beng)

Bengtsson [18] generated his benchmark problem instances in 1982 by assigning rectangular items the nearest integer values to $12r + 1$ as lengths and the nearest integer value to $8r + 1$ as widths, where r is a random number drawn from a uniform distribution on the range $(0, 1)$. The data set contains ten instances, for which the strip width is 25 in five cases, while the strip width is 40 in the remaining cases. These instances have been used by various authors, including Alvarez-Valdés *et al.* [5], Leung *et al.* [116], Yang *et al.* [162], and Wei *et al.* [158] to test the relative performances of algorithms designed for the SPP, and were obtained from [153]. Optimal solutions are known for six of these ten instances.

5.2.3 The Instances of Beasley (gcut and ngcut)

In 1985, Beasley generated two sets of instances to test the relative performances of some packing algorithms. Beasley's first set of benchmark problem instances (denoted by gcut) [14] was generated as a test case for the unconstrained two-dimensional guillotine cutting problem. These instances were created in a manner similar to that adopted by Christofides and Whitlock [40], described in §5.2.1, but with different height and width distributions. The height of a particular rectangular item is an integer taken from a uniform distribution on the range $[h(R_0)/4, 3h(R_0)/4]$ and the corresponding width is taken from a uniform distribution on the range $[w(R_0)/4, 3w(R_0)/4]$.

The second set of benchmark problem instances (denoted by ngcut) [15] was generated imposing the same restrictions as implemented by Christofides and Whitlock, except that the height of a rectangle is an integer drawn from a uniform distribution on the range $[1, h(R_0)]$.

The first data set contains twelve instances for which an optimal solution is known in each case, except for one instance, while the second data set comprises thirteen instances for which optimal solutions are known only for two of the instances. These benchmark instances have been employed by Alvarez-Valdés *et al.* [5], Leung *et al.* [116], Yang *et al.* [162], and Wei *et al.* [158] in the context of the SPP, and were obtained from [153].

5.2.4 The Instances of Berkey and Wang (bwmv)

In 1987, Berkey and Wang [21] generated a set of benchmark instances in order to study the relative performances of their packing algorithms. The problem instances were originally proposed for the 2D bin packing problem and were adapted to the 2D SPP by taking the strip width equal to the bin width while disregarding the other bin dimension. The data set consists of six classes, each comprising five subclasses which contain ten instances each. The instances of a subclass match in terms of bin width and number of items.

The widths and heights of the items were randomly generated from a uniform distribution of integer values. The dimensions of the items in the first and second classes were generated in the range $[1, 10]$, while the range for the third and fourth classes was $[1, 35]$ and the range for the fifth and sixth classes was $[1, 100]$. Items in the six classes are to be packed into strips of widths 10, 30, 40, 100, 100, 300, respectively. These data instances were downloaded from [24] and optimal solutions are not known for all 200 instances.

5.2.5 The Instances of Dagli, Poshyanonda and Ratanapan (DP)

Dagli and Poshyanonda [48], as well as Ratanapan and Dagli [138], provided no details about the method of generation of the problem instances they published in 1997 and no optimal solutions are known for these benchmark instances. These instances have seldom been used in the context of the SPP. Bortfeldt [25] and Ortmann [131] have employed these instances in their computational studies. These data instances were obtained from [60].

5.2.6 The Instances of Burke and Kendall (BK)

In 1999, Burke and Kendall [28] generated a single benchmark instance based on a cutting pattern, a solution returned by the tree-search algorithm proposed by Christofides and Whitlock [40] when applied to one of their test problems, as illustrated in Figure 5.2. This instance consists of convex polygons with a known optimal solution. Burke and Kendall simply doubled the dimensions of the items presented in this figure. This benchmark instance was downloaded from [153].

5.2.7 The Instances of Martello and Vigo (bwmv)

Martello and Vigo [123] generated four classes of benchmark instances in 1998 to expand those proposed earlier by Berkey and Wang [21] (see §5.2.4) for use in the context of bin packing problems. They generated ten problem instances, each containing n items, for the values $n \in \{20, 40, 60, 80, 100\}$ in each class, considering 100×100 bins in each case. The dimensions of the items in each class were selected from four types of items. The first type of item was generated choosing a rectangle height from a uniform distribution on the range $[1, 50]$, and a rectangle width from the range $[67, 100]$. For the second type of items, heights were selected from a

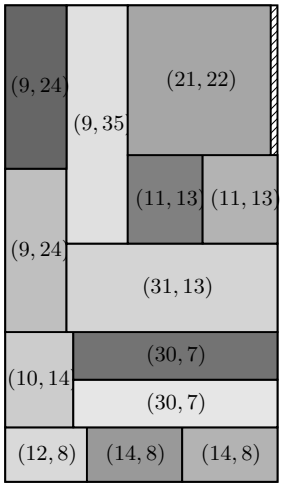


FIGURE 5.2: *The cutting pattern employed by Burke and Kendall [28] when generating their benchmark instance. Entries in parentheses represent the coordinates of the rectangle with the first entry indicating its width and the second entry indicating its height.*

uniform distribution on the range $[67, 100]$ and widths from a uniform distribution in the range $[1, 50]$. The heights and widths of the third type of item were both selected from a uniform distribution on the range $[50, 100]$. The final type of item had both heights and widths selected from a uniform distribution on the range $[1, 50]$.

Each item of an instance of class $k \in \{1, 2, 3, 4\}$ was then of type k with probability 70% and of the remaining types with probability 10% each. These benchmark instances were adapted for the 2D SPP by taking the strip width to be 100 throughout. These instances have been used by Bortfeldt [25], Alvarez-Valdés *et al.* [5], and Wei *et al.* [158], and were obtained from [24]. Optimal solutions are not known for all 200 instances.

5.2.8 The Instances of Hifi (SCPL)

In addition to the set of SPP instances denoted by SCP and described in §5.1.2, Hifi also created a second set of SPP instances (denoted by SCPL) in an attempt to test the effectiveness of his algorithm [83]. The set SCPL was generated in a similar manner as the set SCP, but contains larger instances. These instances have seldom been used in the context of the strip packing problem. Bekrar and Kacem [16], as well as Ortmann [131], have employed them in their computational studies. These data sets were obtained from [82] and optimal solutions are not known for all 9 instances.

5.2.9 The Instances of Wang and Valenzuela (Nice and Path)

In 2001, Wang and Valenzela [156] proposed a recursive process for generating a variety of data sets that can be employed to evaluate the relative performances of cutting and packing algorithms. They designed the procedure in such a way that it can be modified to satisfy user-specified parameters for controlling the height-to-width ratios of the rectangular items. Four different problem generator algorithms were thus developed to create problem instances, with optimal solutions known in all cases.

The first, basic procedure consists of recursively cutting an initial large rectangle into smaller rectangles, placing no restrictions on the relative height and width of each rectangle. At each

step, a slicing position is chosen and the slicing is performed by applying vertical or horizontal cuts along the chosen position. The second algorithm is based on the first one, but additionally respects an *aspect ratio* constraint. At the beginning of the generation process, the range $[1/\rho, \rho]$ is adopted for the aspect ratio of an item, where ρ is a real number larger than 2. The input stock rectangle must satisfy an initial condition based on the value of the parameter ρ. At each iteration, a subrectangle is selected randomly and a slicing direction, horizontal or vertical, is determined. Having chosen the direction in which to slice, an appropriate cutting position is randomly selected and the rectangle is sliced accordingly. These slicing steps are performed repeatedly until the required number of rectangles have been generated.

The third generator algorithm may be employed to control the areas of the rectangles produced. An *area ratio* constraint must be satisfied according to this procedure. That is, the ratio of the areas of any two rectangles in the data set must fall in a given interval $[1/\gamma, \gamma]$, where $\gamma \geq 2$ is again a real number specified by the user. The generation procedure follows the same steps as in the second algorithm and terminates when the required number of rectangles have been generated.

The last algorithm is a combination of the second and third algorithms described above, and may be used to control both the aspect ratios and the area ratios of the rectangles generated.

Half of Wang and Valenzela's benchmark instances were generated by means of the first algorithm described above, which they called the "pathological set" or "path set" for short, while the other half were generated by means of the fourth algorithm (the "nice" set). The values $\rho = 4$ and $\gamma = 7$ were selected to generate these data sets. Each rectangle generated therefore has an aspect ratio in the interval $[0.25, 4]$, and the area of the largest rectangle is no larger than 7 times the area of the smallest. In this manner they generated benchmark problem instances containing rectangles that are either vastly different (path set) or fairly similar (nice set).

Each of these two sets consists of 50 categories, each comprising eight instances with different numbers of items. All these benchmark instances are destined for a strip of width 100 and have an optimal packing height of 100. One significant difference between these data sets and the others described in this section is that the item dimensions are all real numbers, while the other benchmark instances contain items with integer-valued dimensions. These benchmark instances were obtained from [153].

5.2.10 The Instances of Bortfeldt and Gehring (AH)

In 2006, Bortfeldt and Gehring [27] introduced a random number-based problem generator for generating large benchmark instances for the 2D SPP with rectangular items. To generate a problem instance, they enforced certain restrictions with respect to the underlying aspects of the instance itself. These aspects consist of four parameters: A *maximum aspect ratio*, a *maximum area ratio*, a *heterogeneity ratio*, and a *width ratio*. The maximum aspect ratio of all items of an instance is defined as the largest of all items' aspect ratios. The maximum area ratio of all pairs of items of an instance refers to the maximum value of the area ratios of all pairs of items in the data instance. The third factor represents the ratio of the number of types of items, where two items belong to the same type if they have identical smaller and larger side dimensions, to the total number of items in an instance. The width ratio may be described as the ratio of the strip width to the mean value of all (smaller and larger) side dimensions in an instance.

The problem generator takes as inputs the width of the strip W, the number of items n to be generated, the number of types of items nt, and two parameters δ and ρ which are boundary values for the width ratio and the maximum aspect ratio, respectively. The procedure starts by

calculating an interval of item dimensions by means of a pre-defined formula based on the input values. It then successively generates nt random pairs of dimensions within the interval. The procedure thereafter executes a loop to spread the nt dimensions thus obtained evenly over the required total number of items. It stops when the number n is reached.

According to this procedure, a pre-determined heterogeneity ratio, based on the values of n and nt, is achieved for the instance generated. In addition, the respective width ratio is approximately δ. Finally, the maximum aspect ratio and the maximum area ratio of the instance generated are bounded from above by the parameters ρ and ρ^2, respectively.

For a fixed value of the strip width, $W = 1\,000$, and $n = 1\,000$ of items to be generated, Bortfeldt and Gehring established a total of 360 benchmark instances. These benchmark data consist of 12 subsets containing 30 instances each. Different parameter values were employed by the authors to generate instances for each subset. Details of these value combinations may be found in [27]. These data sets were downloaded from [117] and optimal solutions are not known for all 360 instances.

5.2.11 The Instances of Leung and Zhang (Zdf)

Leung and Zhang [115] generated nine large problem instances in order to test the relative performances of their packing algorithms in 2011. These benchmark instances were created in such a way that each instance consists of a combination of existing zero-waste instances and non-zero-waste instances. The zero-waste instances include the beng and gcut instances of Bengtsson [18] and Beasley [14], respectively. The non-zero-waste instances contain the N test set of Burke *et al.* [29], and the CX data instances of Pinto and Oliveira [133].

The first instance, zdf1, for example, consists of four copies of beng1 (the first instance of the beng data set described in §5.2.2) and one copy of N12 (the twelfth instance of the N test set described in §5.1.2), as illustrated in Figure 5.3. The width of zdf1 is equal to the width of N12, $W = 100$, and the total number of items involved in this instance is $n = 20 \times 4 + 500 = 580$, which is the sum of the number of items involved in each instance copy. These instances have been used by Leung *et al.* [116], Yang *et al.* [162], and Wei *et al.* [158], and were obtained from [60]. Optimal solutions are known for only five of the instances.

FIGURE 5.3: *The first instance, zdf1, of Leung and Zhang [115]. N12 is the twelfth instance of the N test set of Burke* et al. *[29], while beng1 is the first instance of the beng data set of Bengtsson [18].*

5.3 Chapter Summary

A brief description of the benchmark instances employed in a comparative study later in this dissertation was provided in this chapter. The class of zero-waste problem instances was discussed first. The respective problem instances were presented in chronological order, starting with the instances of Jakobs [99] in §5.1.1. This was followed by the instances of Hifi [81] and

Authors	Test set	Year	Reference	Number	Class	Comments
Christofides & Whitlock	cgcut	1977	[40]	3	Non-zero-waste instances	1 optimal solution known
Bengtsson	beng	1982	[18]	10	Non-zero-waste instances	6 optimal solutions known
Beasley	ngcut	1985	[15]	13	Non-zero-waste instances	2 optimal solutions known
Beasley	gcut	1985	[14]	12	Non-zero-waste instances	11 optimal solutions known
Berkey & Wang	bwmv	1987	[21]	300	Non-zero-waste instances	No optimal solutions known
Jakobs	J	1996	[99]	2	Zero-waste instances	Both optimal solutions known
Dagli & Poshyanonda	DP	1997	[48]	11	Non-zero-waste instances	No optimal solutions known
Martello & Vigo	bwmv	1998	[123]	200	Non-zero-waste instances	No optimal solutions known
Ratanapan & Dagli	DP	1998	[138]	1	Non-zero-waste instances	Optimal solution unknown
Hifi	SCP	1998	[81]	25	Zero-waste instances	All optimal solutions known
Burke & Kendall	BK	1999	[28]	1	Non-zero-waste instance	Optimal solution known and guillotineable
Hifi	SCPL	1999	[83]	9	Non-zero-waste instances	No optimal solutions known
Babu and Babu	babu	1999	[8]	1	Zero-waste instance	Optimal solution known
Hopper & Turton	C	2001	[91]	21	Zero-waste instances	All optimal solutions known and 14 guillotineable
Wang & Valenzuela	nice & path	2001	[156]	480	Non-zero-waste instance	All optimal solutions known and guillotineable
Hopper & Turton	NT(n) & NT(t)	2002	[92]	70	Zero-waste instances	All optimal solutions known and 35 guillotineable
Burke *et al.*	N	2004	[29]	12	Zero-waste instance	All optimal solutions known and guillotineable
Pinto & Oliveira	CX	2005	[133]	7	Zero-waste instance	All optimal solutions known
Bortfeldt & Gehring	AH	2006	[27]	360	Non-zero-waste instances	No optimal solutions known
Imahori & Yagiura	IY	2010	[94]	170	Zero-waste instances	All optimal solutions known
Leung & Zhang	zdf	2011	[115]	16	Non-zero-waste instances	5 optimal solutions known

TABLE 5.1: *The 1718 benchmark problem instances used to evaluate the performances of the SPP algorithms considered in this dissertation.*

of Babu [8] in §5.1.2 and §5.1.3, respectively. The three data sets of Hopper and Turton [91, 92] were presented next in §5.1.4, namely NT(n), NT(t), and C. These data sets were followed by descriptions of the N benchmark instances of Burke *et al.* [29] in §5.1.5 and the CX instances of Pinto and Oliveira [133] in §5.1.6. The final zero-waste problem instances considered in this dissertation are the IY instances of Imahori and Yagiura [94], described in §5.1.7.

The class of non-zero-waste problem instances was further discussed in §5.2. The respective problem instances were also presented in chronological order, starting with the instances of Christofides and Whitlock [40] in §5.2.1. This was followed by the beng instances of Bengtsson [18] in §5.2.2, and the gcut and ngcut instances of Beasley [14, 15] in §5.2.3. The data sets of Berkey and Wang [21], Dagli and Poshyanonda [48], Ratanapan and Dagli [138], Hifi [83], Burke and Kendall [28], and Martello and Vigo [123] were presented next in §5.2.4, §5.2.5, §5.2.6, §5.2.7, and §5.2.8, respectively. The Nice and Path instances of Wang and Valenzuela [156], the AH instances of Bortfeldt and Gehring [27], and the zdf instances of Leung and Zhang [115] were finally presented in §5.2.9, §5.2.10, and §5.2.11, respectively.

The characteristics of the data sets reviewed in this chapter are summarised in Table 5.1. These data instances may be downloaded from the EURO Special Interest Group on Cutting and Packing (ESICUP) repository [60], the online repository of Lijun and Wenbin [117], the library of instances maintained by Hifi [82], the library of instances of the Operations Research Group Bologna [24], the repository for cutting and packing problems of Imahori and Yagiura [95], and the repository for strip packing problems published online by Van Vuuren and Ortmann [153].

CHAPTER 6

Cluster Analysis: A review

Contents

A brief literature review is provided in this chapter with respect to cluster analysis. This consists of a brief description of the different clustering techniques and methods of clustering validation employed in this dissertation. An overview of the topic of clustering is first presented in §6.1. This is followed by a presentation of the most prominent examples of clustering algorithms in §6.2. Thereafter, various clustering validation measures are described in some detail in §6.3. Finally, a brief summary of the chapter contents is provided in §6.4.

6.1 Overview of Clustering Methods

The purpose of this section is to provide an overview of cluster analysis. A brief background on the subject is first presented. This is followed by a discussion on the basic processes required when performing a cluster analysis. Finally, four well-known distance measures are described.

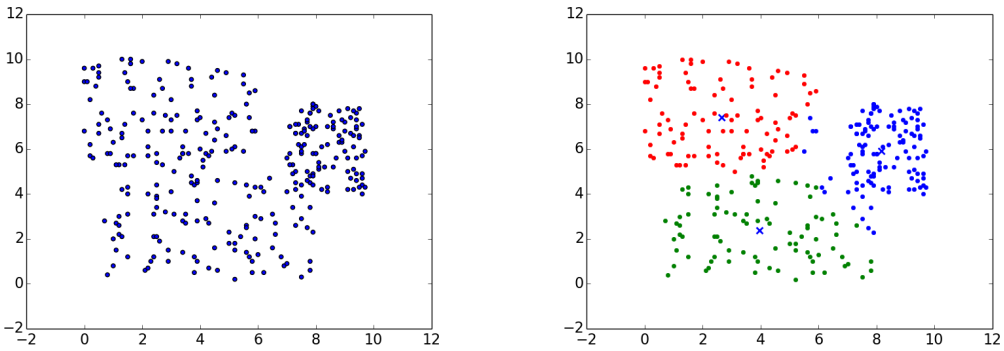
6.1.1 Background

Clustering, also referred to as *cluster analysis*, may be described as the classification of objects (data items or observations) into groups. Objects that differ in insignificant details are grouped together to form a cluster. Jain *et al.* [98] define cluster analysis as "the organization of a collection of patterns (usually represented as a vector of measurements, or a point in a multidimensional space) into clusters based on similarity. Intuitively, patterns within a valid cluster are more similar to each other than they are to a pattern belonging to a different cluster." Berkhin [22] explains that "clustering is a division of data into groups of similar objects. Each group, called cluster, consists of objects that are similar between themselves and dissimilar to objects of other groups."

The clustering problem is old, its lineage dating back to Aristotle [78]. It is one of the most widely used techniques for exploratory data analysis, and has been addressed in many contexts and by researchers in many disciplines, with applications ranging from statistics and computer science to biology, the social sciences, and psychology [34, 98, 161]. Accordingly, several terms have been utilised in the literature for this class of techniques, such as *unsupervised learning* [97], *numerical taxonomy* [145], *vector quantization* [129], and *learning by observation* [124].

General references and important survey papers on clustering include [1, 22, 78, 79, 80, 98, 110, 161]. Jain *et al.* [98] reviewed clustering methods from a statistical pattern recognition point of view and described applications in image segmentation, object recognition, and information retrieval. Kolatch [110] and He [80] also investigated the application of clustering algorithms to spatial database systems and information retrieval, respectively. Hansen and Jaumard [78] presented a survey on the topic under a mathematical programming scheme, while Berkhin [22] reviewed the topic in the context of data mining.

The ultimate goal of clustering is to discover the natural groupings of a set of data points, such that data items belonging to the same group share similar characteristics or features. Given a representation of n data points, the objective of clustering is to find k groups of points, based on some similarity measure, such that the within-group similarities are large while the inter-group similarities are small [96]. An example of clustering is shown in Figure 6.1. A significant challenge is usually to identify or develop an appropriate clustering algorithm that will generate natural groupings into clusters (Figure 6.1(b)) of the given input data (Figure 6.1(a)).



(a) *Input data* (b) *Desired clustering*

FIGURE 6.1: *An example of a clustering procedure applied to a set of data points.*

A well-defined clustering should exhibit the properties of *external isolation* and *internal cohesion* [46]. The phrase *external isolation* stresses the fact that similar items should not be placed

in different clusters and that a discontinuity should be observable between clusters. *Internal cohesion*, on the other hand, emphasises the fact that items belonging to the same cluster should be similar to each other, at least within the local metric. Clusters furthermore possess a number of properties, the most important of which are *density*, *variance*, *shape*, and *separation* [80, 145]. Density refers to the degree of compactness of a cluster in the feature space of the data, while variance refers the degree of dispersion of the data points in this space from the centre of the cluster. Shape is furthermore related to the arrangement of the data points in the feature space, while separation is the degree to which clusters overlap or lie apart in this space.

6.1.2 Clustering Process

Typical cluster analysis involves the following steps: *Feature extraction or selection*, *clustering algorithm design*, *clustering output assessment*, and *results interpretation*, as illustrated in Figure 6.2. Feature extraction or selection is a preprocessing step, whereby a set of features is identified for use during the clustering process. This step is required to enhance the quality of the eventual clustering. As pointed out by Jain *et al.* [98] as well as Aggarwal and Reddy [1], feature selection refers to the process of choosing distinguishing features from an original set of candidates, while feature extraction employs some transformation in order to produce prominent novel features. These procedures might involve a trial-and-error approach, where various subsets of features are selected and the resulting clusters are evaluated by means of certain validity indices. More information on feature selection may be found in [1, 98, 161].

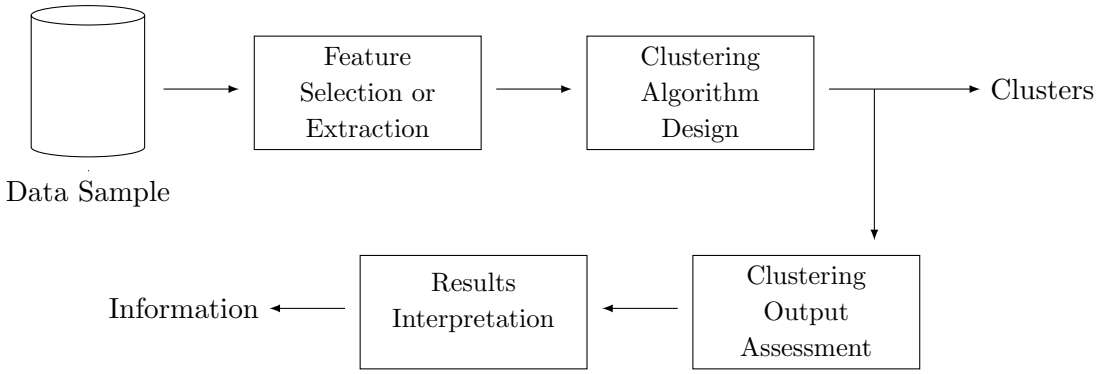


FIGURE 6.2: *A schematic representation of the process of clustering data.*

Clustering algorithm design encompasses the selection of a proximity or distance measure, and the choice of an appropriate clustering algorithm for subsequent use. Data points are to be grouped together based on their similarities. Therefore, the proximity measure affects the construction of clusters and has to be chosen carefully. It is noted that almost all clustering algorithms are explicitly or implicitly dependent on some definition of a proximity measure [161]. An abundance of clustering algorithms has furthermore been proposed in the literature for solving different types of clustering problems in a variety of different fields. There is, however, no clustering algorithm that is generally applicable to all types of clustering problems. It is, therefore, important to investigate the problem at hand properly in order to select an appropriate clustering method.

Clustering output assessment refers to the process of evaluating the clustering results derived from the selected and employed algorithms for validation purposes. Usually, different clustering techniques result in different clusters, and even for the same algorithm, different input parameters typically lead to different cluster results [22, 98]. Effective evaluation or testing criteria are

therefore required for the assessment of the performance of the algorithms considered. A later section of this chapter is devoted to a brief overview of this process.

Results interpretation is the process of extracting information from the clustered data. It involves interpretation of the clustering algorithm output, as well as characterisation of the clustered data based on the features selected. Since the main goal of clustering is to provide the analyst with meaningful insights into the original data and to effectively solve the data interpretation problem at hand, it might also be required to conduct further analyses during this phase in order to ensure reliability of the knowledge extracted.

6.1.3 Popular Distance Measures

Since the notion of similarity is fundamental in clustering analysis, a similarity measure is essential to most clustering techniques, and must be chosen carefully [98, 161]. The selection of a similarity measure is problem-dependent, based on the characteristics of the data features. It is most common to use *dissimilarity* or *distance measures* for clustering purposes, defined on the feature space, in order to evaluate the similarity or closeness between a pair of data points, a data point and a cluster, or a pair of clusters [22, 98, 161]. In this section, the focus falls briefly on popular distance measures employed for data points whose features are all continuous.

Well-known distance measures for continuous features include the *Euclidean*, *Manhattan*, and *Chebyshev* distances. The Euclidean distance between two k-dimensional data points $\boldsymbol{x} = (x_1, \ldots, x_k)$ and $\boldsymbol{y} = (y_1, \ldots, y_k)$ is the square root of the sum of the squares of the differences between corresponding components, expressed mathematically as

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}.$$

The Manhattan distance, also known as the *City block distance*, between the data points $\boldsymbol{x}$ and $\boldsymbol{y}$, is the sum of the absolute differences between their corresponding components, expressed as

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{k} |x_i - y_i|.$$

The aforementioned two distance measures are special cases of the *Minkowski distance*, which is defined as

$$d(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{i=1}^{k} |x_i - y_i|^q \right)^{\frac{1}{q}},$$

for some positive integer q. One drawback of direct use of the Minkowski distance measure is the tendency of a large-scaled feature to dominate the others. In order to avoid this disadvantage, the features of the data points may be normalised or standardised to a common range or variance.

The Chebyshev distance between the data points $\boldsymbol{x}$ and $\boldsymbol{y}$ is the greatest of the absolute differences between their corresponding components, expressed as

$$d(\boldsymbol{x}, \boldsymbol{y}) = \max_i |x_i - y_i|.$$

6.2 Clustering Techniques

Clustering of data may be achieved using a wide variety of methods [1, 22, 98, 161]. These methods differ significantly in their presumed interpretation of what represents a cluster, as well

as in their underlying processes for finding the different clusters. Typical notions of clusters include groups with small distances among the cluster members, dense areas of the data space, and intervals defined by a particular statistical distribution. Understanding the key concept of the various clustering methods available in the literature facilitates a thorough understanding of the difference between the various algorithms on the one hand, and facilitating the possibility of efficiently implementing the algorithms considered, on the other hand.

Berkhin [22] claimed that "categorisation of clustering algorithms is neither straightfoward, nor canonical." Different starting points and clustering criteria lead to different taxonomies of clustering algorithms [1, 22, 96, 98, 161]. A rough, but widely agreed upon, framework for classifying clustering techniques involves partitioning these algorithms into the classes of *partitional methods* and *hierarchical methods*. Partitional clustering algorithms partition data points into sets of disjoint clusters. These algorithms are usually iterative in nature, attempting to determine the correct number of partitions that optimises a certain criterion. Hierarchical algorithms, on the other hand, create clusters recursively by either merging smaller clusters into larger ones or successively splitting large clusters into smaller ones. Other types of clustering techniques include the class of *density-based* clustering methods, in which clusters are defined as connected dense regions in the data points space, and the class of *spectral clustering* techniques, in which data points are partitioned into disjoint clusters according to a specific similarity measure. The most prominent examples of clustering algorithms within each of these categories are briefly described in this section.

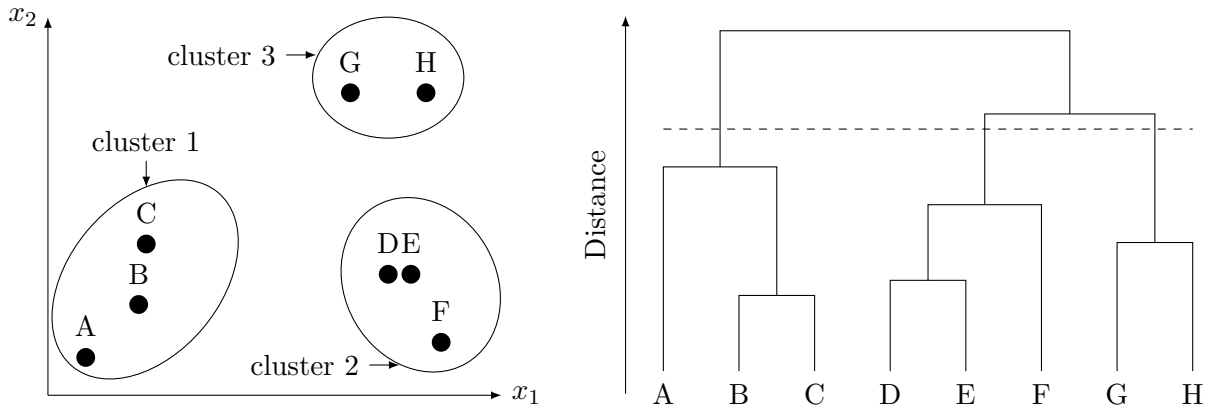
6.2.1 Hierarchical Clustering

Hierarchical clustering algorithms, also known as connectivity-based clustering algorithms, organise data into hierarchical structures by connecting data points to form clusters according to some proximity measure. These algorithms do not provide a unique partitioning of the data set, but rather generate an extensive hierarchy of clusters which can be merged at certain distances. Accordingly, a hierarchical algorithm yields a *dendrogram* or a *binary tree* representation of the nested grouping of items and similarity or distance levels at which groupings change. The root node of such a dendrogram represents the entire data set, while each leaf node represents a data item. The height of the dendrogram expresses the distance between each pair of clusters. In other words, it indicates the similarity level at which the clusters can be merged. A dendrogram corresponding to the eight points in Figure 6.3(a) is shown in Figure 6.3(b). The dendrogram can be cut at different levels to generate different clusterings of the data. The horizontal cut of the dendrogram indicated by the dotted line in Figure 6.3(b) corresponds to the three clusters of the data points shown in Figure 6.3(a).

Hierarchical clustering algorithms may be classified as *agglomerative techniques* and *divisive techniques*. The former techniques adopt a "bottom-up" approach — each data point initially forms its own cluster, and pairs of clusters are aggregated as one moves up the hierarchy, while the latter class of techniques adopt a "top down" approach and proceed in the opposite way — the entire data set initially forms one cluster, which is partitioned recursively as one moves down the hierarchy. In this dissertation, the focus is on the agglomerative clustering; divisive clustering is not commonly used in practice as it typically requires high computational effort [161].

The general procedure of agglomerative clustering can be stated as follows: Start with N singleton clusters, where N represents the cardinality of the data set. Then compute a proximity matrix[1] containing the distance between each pair of clusters. By using the proximity matrix,

[1]For a data set with N input points, a proximity matrix can be defined as an $N \times N$ symmetric matrix whose $(i,j)^{th}$ element represents the distance measure for the i^{th} and j^{th} points $(i, j = 1, \ldots, N)$.



(a) *Data points* (b) *Corresponding dendrogram*

FIGURE 6.3: *(b) An example of a dendrogram corresponding to the seven points in (a)*

find the most similar pair of clusters and merge these two clusters into one cluster. Thereafter, update the proximity matrix to reflect this change in clusters, and repeat the process until all data elements are in one cluster.

Based on the different notions of distance between two clusters, various agglomerative clustering algorithms can be distinguished. The most popular techniques include *agglomerative single-link* and *agglomerative complete-link* algorithms. According to the single-link methods, the distance between two clusters is determined by the minimum of all pairwise distances between elements in the two clusters. In complete-link clusters, on the other hand, the maximum of the distances between all pairs of elements drawn from the two clusters are taken as inter-cluster distance. More detailed information on the various agglomerative clustering techniques available in the literature may be found in [97, 98, 126, 161].

6.2.2 Partitional Clustering

Unlike hierarchical clustering algorithms, which yield a clustering structure of successive levels of clusters by iterative combinations or segregations of data clusters, partitional clustering algorithms generate a single partition by dividing data into several subsets according to no particular hierarchical structure. Typically, an optimal partition can be found by checking all possible subsets and enumerating them. This brute-force method is, however, computationally infeasible in practice [161]. Therefore, greedy heuristic algorithms in the form of iterative optimisation procedures have been developed instead for finding approximate solutions. As such, most of the partitional algorithms available in the literature produce clusters by optimising some criterion function defined on a subset of the data points or over all the data points [98].

The most widely used criterion function in partitional clustering algorithms is the *squared error criterion*. For a set of data points $\boldsymbol{x}_j \in \mathbb{R}^K$, $j = 1, \ldots, N$, to be grouped into K clusters, the squared error criterion is defined as

$$J = \sum_{i=1}^{K} \sum_{j=1}^{N} ||\boldsymbol{x}_j^{(i)} - \boldsymbol{c}_i||^2,$$

where $\boldsymbol{x}_j^{(i)}$ denotes the j-th data point belonging to the i-th cluster and $\boldsymbol{c}_i$ denotes the centroid of the i-th cluster.

The k-means algorithm is the best-known algorithm employing the squared error criterion. It starts with a random initial partition and computes the means or centroids of the clusters in this partition. After initialisation, it loops between two steps, an assignment step during which each data point is assigned to its nearest centroid, and an update step during which a new centroid is calculated for each cluster. These steps are repeated until some convergence criterion is met (*e.g.* the requirement that the difference between the old and new centroids is less than a predetermined threshold value).

The k-means algorithm is frequently used in practice because it is easy to implement, and its time complexity is linear with respect to the number of data points. This is a considerable advantage in applications involving large data sets and the algorithm provides good results in terms of compactness of clusters. Several drawbacks of this algorithm have also been pointed out in the literature [98, 161]. One such disadvantage is its sensitivity with respect to the selection of the initial partition. The convergence of the algorithm varies as a function of different initial centroids: Incorrect initial partitions may lead the algorithm to converge to a local optimum instead of a global optimum. A general strategy aimed at overcoming this disadvantage is to run the algorithm many times with random initial partitions. Other limitations of the k-means algorithm include its sensitivity to outliers and noise in the data, and its applicability to numerical variables only. As a result, many variations on the k-means algorithm have been proposed to overcome these disadvantages. More detailed information on these variations may be found in [98, 161].

6.2.3 Spectral Clustering

It has been reported that some data sets possess arbitrary non-convex geometric shapes and that the traditional clustering algorithms, such as the k-means algorithm, struggle to achieve good results in such cases [1, 163]. The family of spectral clustering algorithms has been developed to effectively handle such types of data sets. This class of techniques relies on the eigen-structure of a similarity matrix, constructed from the similarity graph between the data points, to partition the data into disjoint clusters with points in the same cluster achieving high pairwise similarity and points in different clusters exhibiting low similarity.

Spectral clustering is performed by first representing the given data points in the form of an undirected, weighted *similarity graph* $G = (V, E)$. Each vertex in this graph represents a data point. Two vertices are joined by an edge if the similarity between the corresponding data points is acceptable according to some pre-defined criterion, and the edge is weighted by the pairwise similarity. Several similarity graph constructions exist in the literature, which all model the local neighbourhood relationships between the data points [1, 155]. An example is the *ϵ-neighbourhood graph*, which attempts to join all points whose pairwise distances are smaller than a pre-selected positive real value ϵ.

Once the similarity graph has been constructed, the next step of the spectral clustering process consists of computing the corresponding graph Laplacian matrices and deriving the respective eigenvectors to define the various connected components of the graph. In other words, the data points are embedded in a space of appropriate dimension, in which a suitable cluster separation is clear by means of the use of the eigenvectors of the graph Laplacian. Upon execution of this step, a classical clustering algorithm, such as the k-means algorithm, is applied to partition the embedding into clusters. Discussions and surveys on the various spectral clustering algorithms available in the literature may be found in [1, 9, 155, 163].

6.2.4 Density-based Clustering

As pointed out in [1, 22, 39], some of the algorithms reviewed above are unable to find clusters of arbitrary shapes and struggle to handle outliers and noise. Furthermore, these algorithms usually require knowledge of a suitable number of clusters *a priori*, which cannot be easily determined for some types of data sets. The paradigm of density-based clustering has been proposed to address these issues, by making no assumption about the number of clusters and simultaneously solving the problem related to outliers and noise. Density-based clustering algorithms apply a local clustering criterion: Clusters are regarded as regions of high data density (areas in the data space in which the points are dense), and which are separated by regions of low data density (noise).

There exist two major approaches towards density-based clustering [22, 161]. The concept of density in the first approach is based on the local distribution of nearest neighbours of data points, while it is based on a general notion of influence functions[2] in the latter approach, which models the influence of a data point on its neighbourhood. In this dissertation, the focus is on the first approach. More specifically, the well-known DBSCAN (Density Based Spatial Clustering of Applications with Noise) algorithm of Ester *et al.* [61] is employed.

The key idea behind the DBSCAN algorithm is that for each member of a cluster, the cardinality of the respective neighbourhood has to exceed a given threshold, *i.e.* the neighbourhood of a given radius ϵ has to contain at least a minimum number $MinPts$ of data points. The main concepts of the DBSCAN algorithm are *density-reachability* and *density-connectivity*. A data point $\boldsymbol{p}$ is density-reachable from a point $\boldsymbol{q}$ if $\boldsymbol{p}$ is within ϵ-neighbourhood of $\boldsymbol{q}$, and $\boldsymbol{q}$ has more than $MinPts$ data points in its ϵ-neighbourhood. Two points $\boldsymbol{p}$ and $\boldsymbol{q}$ are density-connected if there is a point $\boldsymbol{r}$ from which both $\boldsymbol{q}$ and $\boldsymbol{p}$ are density-reachable. The DBSCAN algorithm interprets a cluster as a set of density-connected points that is maximal with respect to density-reachability.

Given the two parameters ϵ and the minimum number of points $MinPts$ required to form a cluster, the DBSCAN algorithm initialises with an arbitrary starting point that has not been visited and extracts all points that are density-reachable from this point with respect to the parameters ϵ and $MinPts$. If there are sufficient neighbours around this point, then the procedure yields a cluster and that point is marked as visited. If not, the point is labelled as noise. If the point is a border point of some cluster, it will later be density-reachable and become a member of that cluster. The procedure is repeated for another unvisited data point, and the algorithm terminates when all data points have been marked as visited.

6.3 Clustering Validation Measures

Halkidi *et al.* [75] claimed that "the majority of the clustering algorithms behave differently depending on the features of the data set and the initial assumptions for defining groups." The clustering result obtained from a particular algorithm can be very different from another one applied to the same data set as their respective input parameters can, for example, affect their execution behaviour significantly. Therefore, the clustering scheme selected requires an evaluation process for assessing its validity. The evaluation of clustering algorithmic results is referred to as *clustering validation* or *cluster validity* assessment.

Generally, clustering validation techniques can be categorised into three classes [75, 149]. The first class, known as *external clustering validation methods*, is based on external criteria. Tech-

[2] An influence function describes the effect of a change in one data point on an estimator. A typical example of an influence function is the Gaussian function.

niques in this class compare the output of a clustering algorithm with an externally known result, such as externally provided class labels. Such techniques can be used to select an appropriate clustering algorithm for a specific data set. Methods in the second class, referred to as *internal clustering validation methods*, are based on internal criteria. In this case, the internal information of the clustering process is used to evaluate the suitability of its output without reference to external information. Techniques in this class can be used to estimate an appropriate number of clusters and also to find an appropriate clustering algorithm for a specific data set. Techniques in the last class, known as *relative clustering validation methods*, are based on relative criteria. Here the basic idea is to evaluate the clustering structure by varying different parameter values for the same algorithm (*e.g.* varying the number of clusters). Techniques in this class are generally used for determining an optimal number of clusters.

A variety of clustering validity measures have been proposed in the literature within each of the above classes [49, 54, 73, 74, 75, 76]. In this dissertation, the focus is on internal clustering validation measures. These measures are based on the *compactness* and the *separation* of the cluster partitions. The compactness measures assess how close the members of the same cluster are to one another. A smaller within-cluster variation is an indicator of good compactness, *i.e.* a good clustering. Indices for evaluating the compactness of clusters are usually based on distance measures, such as the clusterwise-within-average distances between data points. Separation measures, on the other hand, determine how well separated a cluster is from other clusters. There are three common approaches toward measuring the distance between two different clusters [112]: distances between cluster centres, distances between the closest members of the clusters, and distances between the most distant members of the clusters. The most widely used clustering validation indices are reviewed briefly in this section.

6.3.1 The Silhouette Coefficient

The silhouette coefficient [139] measures how appropriately a data point is clustered and estimates the average distance between clusters. It contrasts the average distance between members of the same cluster with the average distance from these data points to members of other clusters. A larger silhouette coefficient value corresponds to better defined clusters.

For a given cluster of data points, the silhouette coefficient is defined as

$$s = \frac{b - a}{\max\{a, b\}},$$

where a represents the mean distance between members of the cluster, and b denotes the mean distance between members of the cluster and all other data points in the next nearest neighbouring cluster. The silhouette coefficient for a set of clusters is taken as the mean of the silhouette coefficients for the various clusters.

6.3.2 The Caliński-Harabasz Index

The Caliński-Harabasz index [33] evaluates clustering performance based on the average between-cluster and within-cluster dispersion. The between-cluster dispersion relates to the dispersion of the clusters among each other. More precisely, it is defined as the sum of squared distances between the centre of each cluster and the centre of the entire data set. Within-cluster dispersion, on the other hand, is defined as the dispersion of the members of the cluster with respect to its centre. The index is larger when clusters are dense and well separated.

For N data points partitioned into k clusters, the Caliński-Harabasz index is defined as

$$s(k) = \frac{\mathrm{Tr}(\boldsymbol{B}_k)}{\mathrm{Tr}(\boldsymbol{W}_k)} \times \frac{N-k}{k-1},$$

where $\mathrm{Tr}(\boldsymbol{B}_k)$ and $\mathrm{Tr}(\boldsymbol{W}_k)$ denote the trace of the between-cluster dispersion matrix $\boldsymbol{B}_k$ and the within-cluster dispersion matrix $\boldsymbol{W}_k$, respectively. These two traces are given by

$$\mathrm{Tr}(\boldsymbol{W}_k) = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} (\boldsymbol{x} - \boldsymbol{c}_i)(\boldsymbol{x} - \boldsymbol{c}_i)^T$$
$$\mathrm{Tr}(\boldsymbol{B}_k) = \sum_{i=1}^{k} n_i (\boldsymbol{c}_i - \boldsymbol{c})(\boldsymbol{c}_i - \boldsymbol{c})^T,$$

where n_i and $\boldsymbol{c}_i$ denote respectively the number of data points in and the centre of the i-th cluster C_i. Furthermore, $\boldsymbol{c}$ denotes the centre of the entire data set.

6.3.3 The Dunn Index

The Dunn index [54] aims to identify dense clusters — with small within-cluster variance, well-separated clusters, and sufficiently large inter-cluster means. The index, denoted here by D, is defined as the ratio between the minimal distance between points of different clusters and the maximal intra-cluster distance, or

$$D = \frac{\min_{1 \le i < j \le k} d(i,j)}{\max_{1 \le i \le k} d'(i)},$$

where $d(i,j)$ denotes the distance between the i-th and j-th cluster centres, and $d'(i)$ represents the within-cluster distance of the i-th cluster. A larger Dunn index value corresponds to better defined clusters.

6.3.4 The Davies-Bouldin Index

The Davies-Bouldin index [49] may be used to infer the appropriateness of a data clustering based on the similarity of the clusters generated. It is defined as the average of the similarity measures between each cluster and its most similar cluster. That is, for each cluster, the largest value of the similarities between that cluster and all other clusters is computed. The Davies-Bouldin index, denoted by DB, is then obtained by averaging all the cluster similarities. That is,

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left\{ \frac{\sigma_i + \sigma_j}{d(\boldsymbol{c}_i, \boldsymbol{c}_j)} \right\},$$

where $\boldsymbol{c}_i$ denotes the centre of the i-th cluster i, σ_i denotes the average distance from all points in the i-th cluster to its centre $\boldsymbol{c}_i$, and $d(\boldsymbol{c}_i, \boldsymbol{c}_j)$ is the distance between the two centres $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$ of the i-th and j-th clusters, respectively ($i \neq j$). Since clusters with high intra-cluster similarities and low inter-cluster similarities are desirable, a small Davies-Bouldin index corresponds to a well-defined cluster structure.

6.4 Chapter Summary

The aim in this chapter has been to review the literature on cluster analysis and to explain the various clustering techniques and clustering validation methods considered and applied in the remainder of this dissertation. A general background on clustering was presented in §6.1.1, and this was followed by a description of various clustering processes in §6.1.2. Three well-known clustering distance measures were reviewed in §6.1.3. Thereafter, four well-known families of clustering techniques were described in §6.2, namely hierarchical clustering, partitional clustering, spectral clustering, and density-based clustering. Finally, a review followed in §6.3 on clustering validation measures. Four of the most widely used internal clustering validation indices were described in that section, namely the Silhouette coefficient, the Caliński-Harabasz index, the Dunn index, and the Davies-Bouldin index.

CHAPTER 7

Clustered Benchmarks

Contents

In this chapter, details of the cluster analysis performed on the SPP benchmark data described in Chapter 5 are presented. The process of data categorisation, *i.e.* the identification of the meaningful factors that best describe the data, is discussed in §7.1. Thereafter, the various steps that were followed during the cluster analysis are presented in §7.2. These include data preparation and visualisation, estimation of the optimal number of clusters, and assessment of the clusters obtained. A description of the clustered benchmarks instances follows in §7.3. Finally, a summary of the chapter contents is provided in §7.4.

7.1 Data Categorisation

As discussed in Chapter 5, a large number of benchmark problem instances from the literature are employed throughout this dissertation to study the performance of the different strip packing algorithms considered. These data instances exhibit a wide range of characteristics since their method of generation differ from one author to the next. As already explained, some data sets were generated by repeatedly cutting an initial rectangle into small pieces (*e.g.* [29, 91, 92, 99]). This method of generation ensures an optimal packing with zero waste. A second group of test instances corresponds to non-zero-waste problems for which optimal solutions involve some wasted regions and not all optimal solutions are known (*e.g.* [14, 15, 18, 156]). The optimal packing heights, in the case of unknown optimal solutions, are estimated by valid lower bounds such as those proposed by Martello *et al.* [122].

It has been reported that the aspect ratios according to which the benchmark instances were generated affect the mean solution quality of packing methods [27]. The results obtained for

the nice set instances of Wang and Valenzela [156] by the solution proposed in [151], for example, always obtained a better mean solution quality than for the path set of the same authors (independent of the problem instance size). The application of the hybrid simulated annealing proposed in [31] to the same instances, on the contrary, achieved the opposite result. As such, it is advocated that the underlying characteristics of the different data sets and their effect on the solution quality achieved by the various packing algorithms should be investigated. It was therefore decided to examine the collected set of SPP instances described in Chapter 5, and to categorise them based on their underlying features. The relevant advantages of such a perspective involve achieving a better understanding of the test problems and a more responsible way of comparing the performances of packing algorithms.

Accordingly, a cluster analysis is performed in this chapter in respect of the available test data. The instances are grouped into different categories based on the following features: The *maximum aspect ratio* of all items of an instance, the *maximum area ratio* of all pairs of items of an instance, the *heterogeneity ratio*, and the *width ratio*.

1. The *maximum aspect ratio* of all items of an instance is determined by

$$\rho = \max \{\rho(i) \mid i = 1, \ldots, n\}.$$

   The parameter n represents the total number of items involved in the given instance. The aspect ratio of an item i is defined as $\rho(i) = d_{max}(i)/d_{min}(i)$, where $d_{min}(i)$ and $d_{max}(i)$ denote its smaller side dimension and larger side dimension, respectively.

2. The *maximum area ratio* of all pairs of items of an instance is defined as

$$\gamma = \max \{\gamma(i, j) \mid i, j = 1, \ldots, n;\ i \neq j\}.$$

   The area ratio of a pair of items i, j is given by $\gamma(i, j) = a(i)/a(j)$, where $a(i)$ denotes the area of item i.

3. The *heterogeneity ratio* is given by $\nu = n_t/n$, where n_t denotes the number of distinct types of items in an instance. Two items are of the same type if they have identical smaller and larger side dimensions.

4. The *width ratio* is determined by $\delta = W/d_{mean}$, where W denotes the width of the strip, and d_{mean} represents the mean value of all (smaller and larger) items' dimensions.

The maximum aspect ratio allows one to gain information on the shapes of the items in an instance, whereas the variety in the sizes of the items in an instance may be deducted from the maximum area ratio. The miscellany of items in an instance may be gauged from the heterogeneity ratio, while the width ratio characterises the mean item width relative to that of the strip.

Each instance is represented by a four-dimensional vector containing as entries the factors described above. These values serve as inputs for the clustering methods employed during the clustering process. A screen shot of a .csv file containing twenty of the data instances, each associated with the corresponding values of the four factors, is shown in Figure 7.1. These factors were calculated based on the above formulas for each instance. A relatively small value of the maximum aspect ratio factor indicates that the respective instance is heavily populated by approximately square shaped items. A larger value of the maximum area ratio factor, on the other hand, implies that the corresponding instance is dominated by items of widely varying sizes. Furthermore, an instance with a relatively large value of the width ratio factor contains a

large number of wide items. Finally, an instance with a value of the heterogeneity ratio factor close to 1 indicates that the dimensions of the items involved in that instance are all different (*i.e.* heterogeneous).

	NameOfFile	NumOfItems	StripWidth	MaxAspectRatio	MaxAreaRatio	WidthRatio	NumTypes	HetRatio
1	2000HopperC3-P2.csv	29	60	6.000000	95.000000	7.565217	7	0.24137931
2	1982Bengtsson10.csv	200	40	12.000000	96.000000	7.133304	50	0.25000000
3	1985BeasleyJORS11.csv	30	1000	2.684211	4.443422	2.130001	30	1.00000000
4	1998Hifi10.csv	4	50	3.000000	3.333333	2.105263	4	1.00000000
5	1998Hifi19.csv	6	5	3.000000	5.000000	1.500000	2	0.33333333
6	1985BeasleyJORS8.csv	50	500	2.394161	7.177057	2.111308	50	1.00000000
7	1977ChristofidesWhitlock3.csv	62	70	1.888889	8.712418	2.653623	22	0.35483871
8	2000HopperT1c.csv	17	200	6.923077	582.750000	4.438642	1	0.05882353
9	1985BeasleyOR4.csv	7	10	9.000000	3.333333	1.505376	2	0.28571429
10	1985BeasleyJORS6.csv	20	500	2.151515	3.826559	1.993223	20	1.00000000
11	1982Bengtsson7.csv	80	40	11.000000	96.000000	6.794055	27	0.33750000
12	2000HopperT3b.csv	29	200	6.913043	53.263158	5.017301	29	1.00000000
13	2000HopperT4c.csv	49	200	5.500000	280.166667	7.237814	8	0.16326531
14	1999Hifi5.csv	30	165	4.000000	39.611111	11.799762	3	0.10000000
15	1985BeasleyJORS13.csv	32	3000	3.784483	4.378837	4.461485	32	1.00000000
16	1982Bengtsson1.csv	20	25	11.000000	32.000000	3.921569	6	0.30000000
17	1985BeasleyJORS9.csv	10	1000	2.255738	2.692852	2.215575	10	1.00000000
18	1999Hifi8.csv	43	475	6.000000	12.750000	35.398614	17	0.39534884
19	2000HopperC4-P1.csv	49	60	7.000000	42.750000	6.720000	11	0.22448980
20	1998Hifi25.csv	15	25	4.000000	12.000000	3.275109	5	0.33333300

FIGURE 7.1: *A screen shot of an Excel table containing twenty of the data instances with their corresponding factor values. The NumOfItems and NumTypes columns contain the number of items and the number of distinct types of items in the instance, respectively.*

7.2 Clustering Process and Assessment

This section contains a description of the different steps that were performed to cluster the benchmark data described in Chapter 5. The first step consisted of preparing the data set. Thereafter, a visual inspection of the data was performed to assess whether the data suggest any obvious meaningful clusters. During the next step, the optimal number of clusters was estimated by means of a variety of indices. This was followed by an evaluation of the performances of the different clustering algorithms so as to choose the best performing one. The resulting clustering output is finally also described.

7.2.1 Data Preparation

A summary of the four feature values over all the benchmark data is presented in Table 7.1. As can be seen in this table, the features exhibit different ranges of values; some features (*e.g.* the width ratio) are about two orders of magnitude larger than the others, while the largest value of the maximum area ratio factor is much larger, about six orders of magnitude larger. In order to avoid dominance of the large values in the clustering results, feature scaling was applied to the data set. Techniques commonly employed for this purpose are *normalisation*, which involves scaling the features' values to within a certain range, and *standardisation*, which transforms the attributes' values to have a zero mean and a unit variance [111]. In this case, the normalisation method was employed to scale the features to within the range [0,1], using the formula

$$\text{ScaledFeature} = \frac{\text{Feature} - \min(\text{Feature})}{\max(\text{Feature}) - \min(\text{Feature})}.$$

Factors	Min	First quartile	Median	Mean	Third quartile	Max
MaxAspectRatio	1.44	3.994	10	28.629	43	322
MaxAreaRatio	1	7.4	66.5	3 151.4	125.4	1 081 080
WidthRatio	0.218	2.727	5.39	9.032	9.99	200.91
HetRatio	0.0004	0.062	0.236	0.466	1	1

TABLE 7.1: *A summary of the four feature values over all the benchmark data.*

7.2.2 Data Visualisation

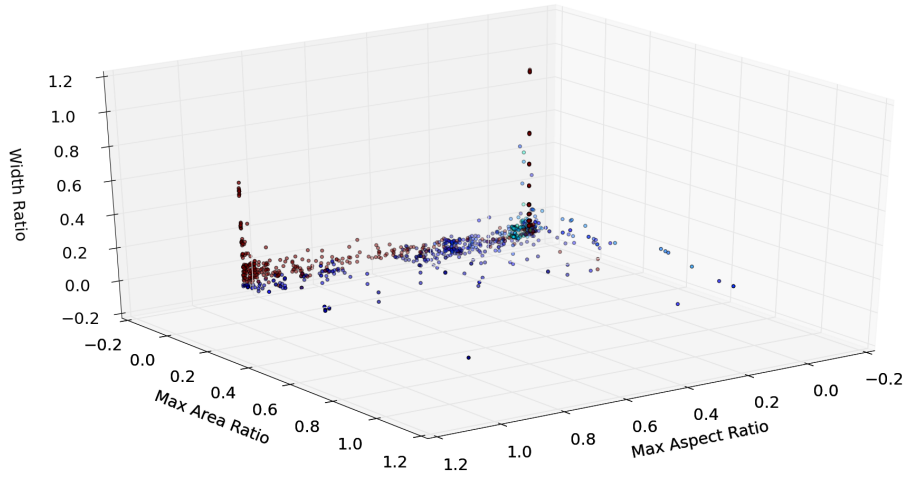
A scatter plot of the data is presented in Figure 7.2(a). As the data contain four variables, it is plotted in three dimensions with the fourth dimension represented by a colour shading. Another way to visualise the data is by means of *Principal Component Analysis* (PCA). PCA is a technique used to emphasise variation and extract patterns in a data set, and is often employed to render data so as to facilitate its exploration and visualisation. PCA involves identification of the principal directions, called "principal components," in which the data vary and reduces the dimensionality of the data by projecting the data onto the principal components identified for a desired dimension, by means of a well-defined mathematical transformation [102, 160]. The **R** function *prcomp()* [7] was employed to compute the PCA of the data set in this study. After performing PCA, the function *fviz_pca_ind()* in the *factoextra* **R** package was utilised to visualise the output. The resulting two-dimensional scatter plot is shown in Figure 7.2(b). The axes in this figure represent a projection that best spreads the data. This visual inspection demonstrates that the data form clusters in some way.

7.2.3 Estimating the Number of Clusters

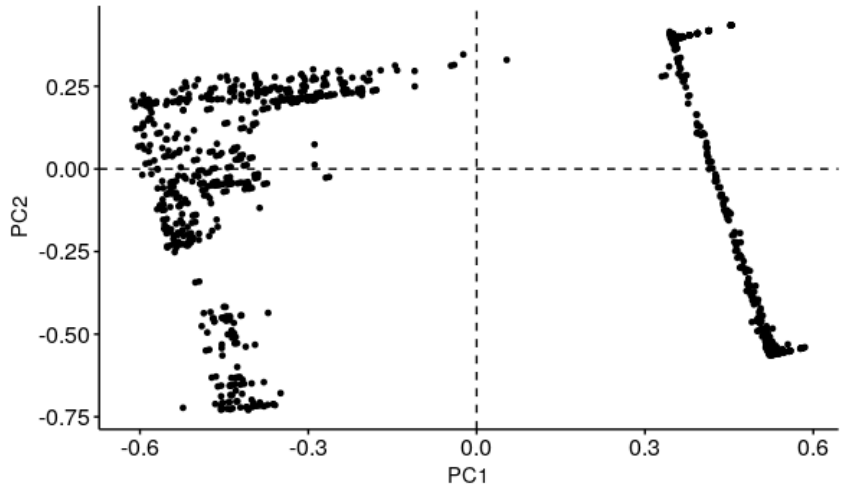
Since the majority of the clustering techniques described in Chapter 6 require a pre-specification of the number of clusters to generate, it is important to determine a suitable number of clusters *a priori*. Various methods have been proposed in the literature for this purpose. Examples of these methods are the *average silhouette method* of Kaufman and Rousseeuw [106] and the *gap statistic method* of Tibshirani *et al.* [150]. By considering different numbers of clusters, the average silhouette method involves computation of the average silhouette of observations for each of these values, returning the one that maximises the average silhouette over this range of cluster numbers as the optimal number of clusters. The gap statistic, on the other hand, involves a comparison of the total within-cluster variation for different numbers of clusters with their expected values under some condition, and the optimal number of clusters is estimated as the value that maximises the gap statistic.

Various indices have also been proposed in the literature for suggesting a suitable number of clusters in a data set [73, 74, 125]. Milligan and Cooper [125], for example, examined thirty indices on a simulated data set, Halkidi *et al.* [73] proposed an index that is based on the concepts of average scattering for clusters and the total separation between clusters, while Halkidi and Vazirgiannis [77] introduced an index based on the criteria of compactness and separation between clusters. In 2014, Charrad *et al.* [36] developed an **R** package, called *NbClust*, which computes thirty indices for estimating a sensible number of clusters in which to partition the data set. With a single function call, it computes all the indices and determines the relevant number of clusters accordingly. It also provides options to the user in respect of varying the

number of clusters, distance measures, and clustering methods in order to obtain a desirable clustering scheme.



(a) *Scatter plot of the data*



(b) *PCA of the data*

FIGURE 7.2: *Visual inspection of the data. (a) A scatter plot of the data in which the fourth variable (heterogeneity factor) is represented by a colour shading. (b) A PCA of the data — the axes, labelled PC1 and PC2, indicate respectively the first and the second principal components and represent a projection that best spreads the data.*

The package *NbClust* was employed in this study to estimate an appropriate number of clusters in which to partition the benchmark data. A histogram of the distribution of the output is shown in Figure 7.3. The majority of the indices suggested four as the best number of clusters.

7.2.4 Choosing the Best Clustering Algorithm

In order to generate a sound data clustering output result, a limited computational study was conducted by evaluating different clustering algorithms with respect to a set of validation measures. Four popular clustering techniques, the k-means algorithm (§6.2.2), the agglomerative

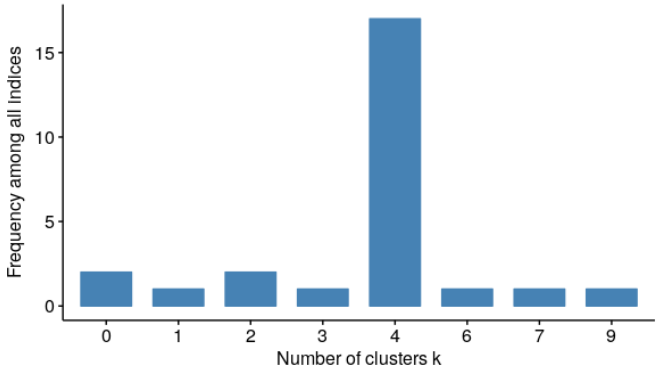


FIGURE 7.3: *Histogram of the distribution of the results obtained when estimating the number of clusters in which to partition the benchmark data of Chapter 5 by means of the* NbClust *function [36]. According to the majority rule, the preferred number of clusters is clearly 4.*

method (§6.2.1), the DBSCAN algorithm (§6.2.4), and the spectral clustering algorithm (§6.2.3), were considered for this purpose. Within the agglomerative technique, the "ward" linkage criterion, which minimises the sum of squared differences within all clusters, was employed. It was found, according to a preliminary comparative pilot study, that this method yields more promising results than the complete and average linkage criteria when applied to the benchmark data of Chapter 5. The ward method was therefore implemented in the agglomerative clustering algorithm in this study.

In the DBSCAN algorithm, the parameters ϵ and $MinPts$ were set equal to 0.25 and 100, respectively. Since the values of these two parameters can significantly affect clustering performance of the algorithm, care must be taken in determining their values. Several techniques have been proposed in the literature for this purpose [59, 105, 141]. The method presented in [141] was implemented in this study. The value for ϵ was chosen by employing a d-distance graph, plotting the distance to the d-th nearest neighbour, for all the data points, and for a given value of d[1]. The sharp change in this plot represents a suitable value of ϵ. After determining the value of ϵ, the number of data points in the ϵ-neighbourhood of every point in the data set was calculated one by one. The value of $MinPts$ was determined by the average of all these number of data points.

The solution quality achieved by these clustering algorithms was assessed by means of the four validation measures, the Silhouette coefficient, the Caliński-Harabasz index, the Dunn index, and the Davies-Bouldin index, all described in §6.3. In Table 7.2, the corresponding comparative result is reported. From this table, it is clear that the k-means algorithm yields larger CH and Silhouette values than the other methods. The corresponding Dunn index is comparatively small, and the value of the DB index is also comparatively large. As explained in §6.3, this result indicates that the k-means clustering algorithm produces a better clustering output than the other techniques. The k-means algorithm was therefore selected as clustering method in this study.

7.2.5 Clustering Output

A two-dimension plot of the clustering output is provided in Figure 7.4. The first cluster contains 321 instances, the second cluster 740 instances, and the third and fourth clusters contain 251 and 406 instances, respectively.

[1]The larger the data set, the larger the value of d should be chosen.

ClusteringAlgo	CH index	Dunn index	DB index	Silhouette coeff
k-means	2383.861	0.0429	0.3834	0.6743
Hierarchical	2226.81	0.0129	0.3604	0.61743
Spectral	2247.143	0.0135	0.508	0.6506
DBSCAN	2373.79	0.0478	0.4569	0.5786

TABLE 7.2: *Performance evaluation of the different clustering algorithms in respect of the benchmark data of Chapter 5, based on the CH, Dunn, DB, and Silhouette indices.*

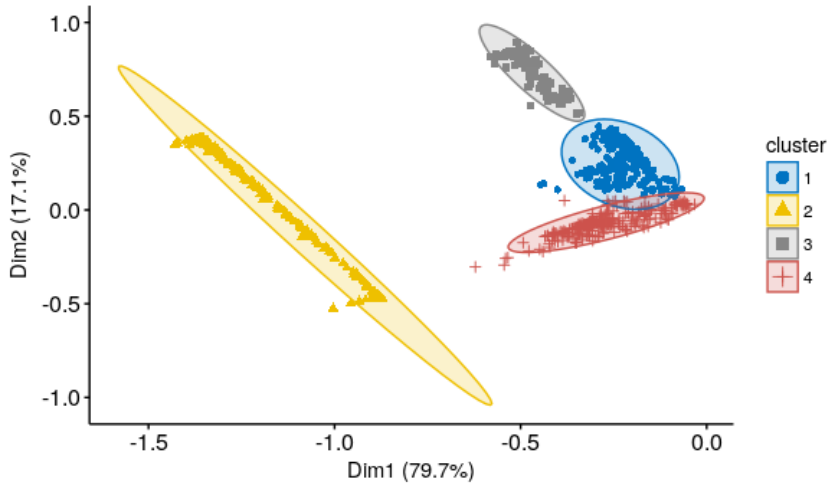


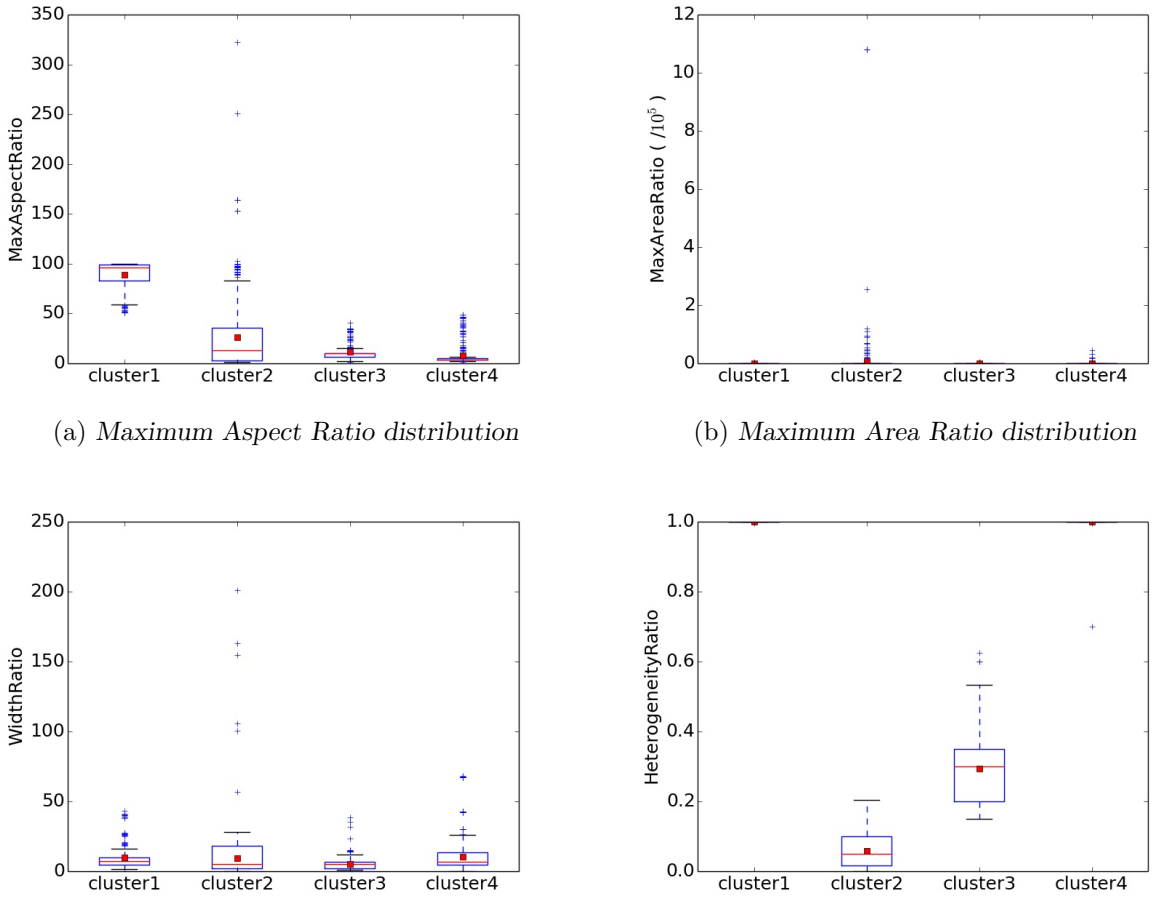
FIGURE 7.4: *Two-dimensional plot of the clustering result obtained by applying the k-means algorithm to the benchmark data of Chapter 5, with $k = 4$, returned by means of Principal Component Analysis. The axes, labelled Dim1 and Dim2, denote the first and second principal components and represent a projection that best spreads the data.*

7.3 Characteristics of the Clustered Data

Boxplots of the distribution of each factor is shown in Figure 7.5, for all four clusters of Figure 7.4. Table 7.3 contains a statistical summary of these boxplots and the underlying characteristics of each cluster.

Cluster 1 is mainly populated by instances with items of an elongated rectangular shape, either long and flat, or tall and thin (the maximum aspect ratio factor lies within a relatively large range of values). The dimensions of items in each instance within this first cluster are all different, *i.e.* no two items are of the same type (the value of the heterogeneity ratio factor is equal to 1 for all instances in this cluster). Furthermore, the range of values of the respective maximum area ratio is within the interval [8.21, 44 605], with a relatively large mean value, indicating that Cluster 1 contains predominantly items of widely varying sizes. The corresponding width ratio value, presented in Table 7.3 and illustrated in Figure 7.5(c), suggests that instances in Cluster 1 contain a large number of narrow items. Most of the instances of the pathological-set of Valenzuela and Wang [151], and some of the instances generated by Berkey and Wang [21] and by Martello and Vigo [123], belong to Cluster 1. An illustrative example is shown in Figure 7.6(a), which is a packing returned by the ISA algorithm when applied to the Path49 instance of Valenzuela and Wang [151].

The second cluster comprises instances with relatively large values of the maximum aspect ratio and the maximum area ratio (that is, items contained in these instances are vastly different with elongated rectangular shapes). Moreover, the value of the heterogeneity ratio for the instances in this cluster is close to zero, indicating that the instances in Cluster 2 are predominantly homogeneous. With regard to the range of values of the width ratio factor, one can say that instances in this second cluster each contains a large number of wide items. The instances in [21, 123], and the AH-set of Bortfeldt and Gehring [27] populate this cluster. Figure 7.6(b) is an illustrative example of the instances in this cluster. The packing in Figure 7.6(b) was generated by the ISA algorithm when applied to the NT(t6b) instance of Hopper and Turton [92].



(a) *Maximum Aspect Ratio distribution*

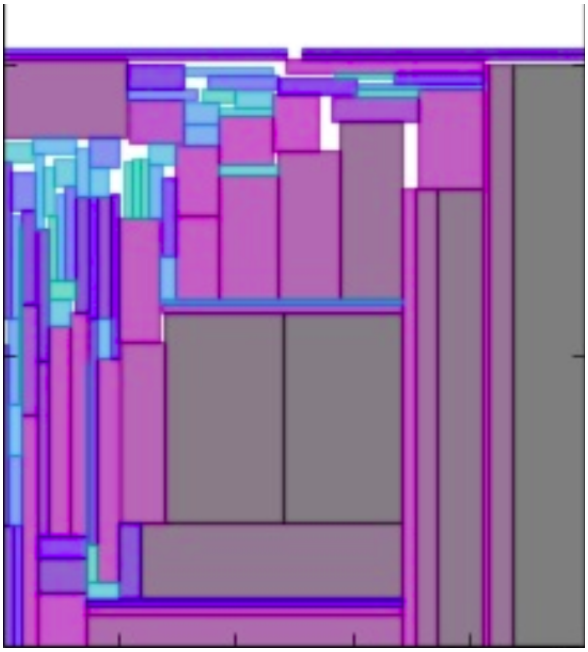
(b) *Maximum Area Ratio distribution*

(c) *Width Ratio distribution*

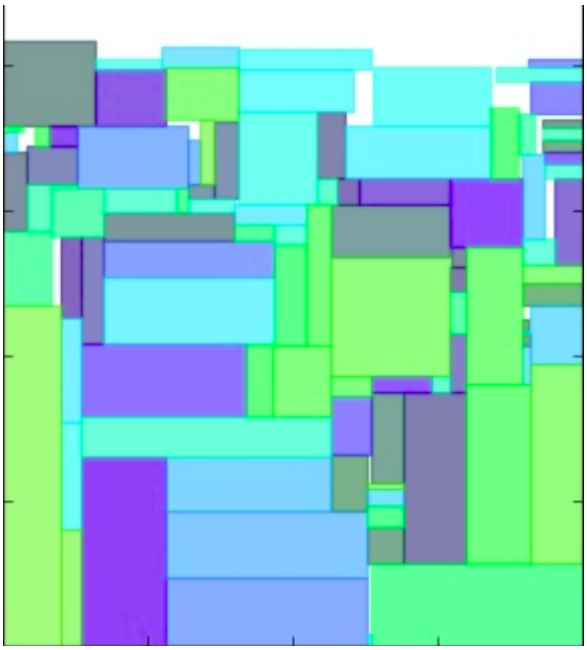
(d) *Heterogeneity Ratio distribution*

FIGURE 7.5: *Boxplots of the distribution of each factor for the four clusters.*

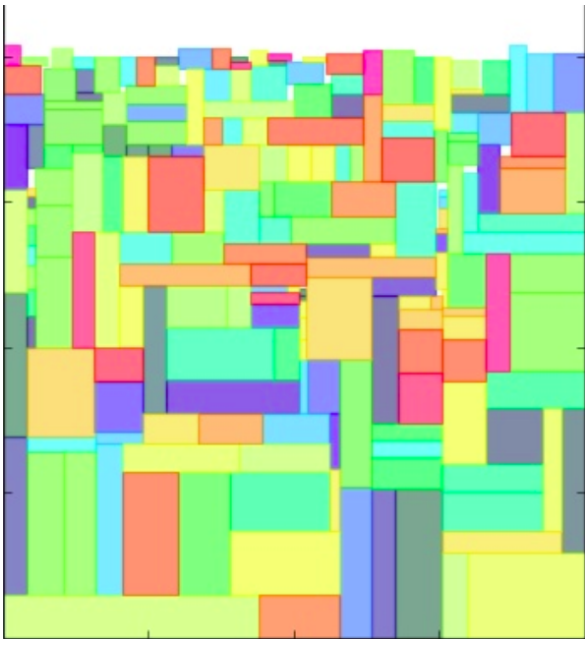
In contrast, instances that belong to Cluster 3 are dominated by square items (the maximum aspect ratio lies within a relatively small range of values), which are relatively small (the width ratio factor also lies within a small range of values). Some of the items in the instances are of the same type (the value of the heterogeneity ratio factor for the instances in the cluster is close to zero). In addition, instances belonging to this cluster are dominated by equally sized items (the respective maximum area ratio lies within a small range of values). Examples of instances in this third cluster include the C-set of Hopper and Turton [91], and the gcut instances of Beasley [14]. Figure 7.6(c) contains an example of instances in Cluster 3. The packing in Figure 7.6(c) was generated by the ISA algorithm when applied to the C6-P1 instance of Hopper and Turton [91].



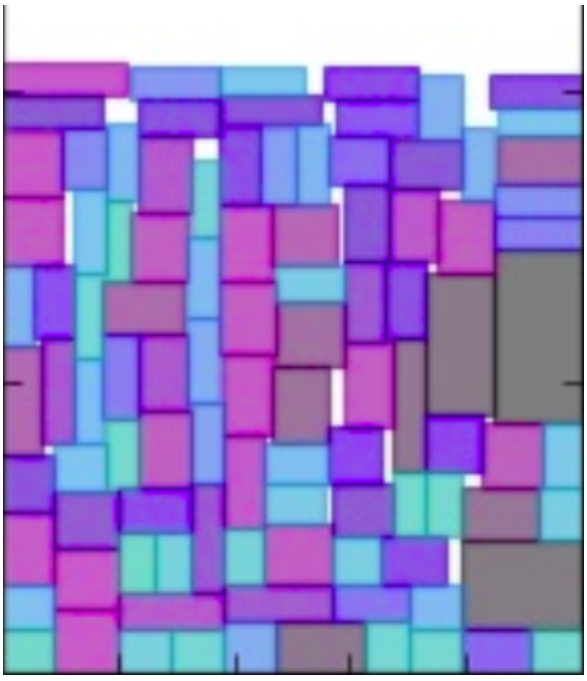
(a) *Path49 with* $n = 100$ *(Cluster 1)*



(b) *T6b with* $n = 97$ *(Cluster 2)*



(c) *C6-P1 with* $n = 97$ *(Cluster 3)*



(d) *Nice50 with* $n = 100$ *(Cluster 4)*

FIGURE 7.6: *Examples of instances belonging to each of the four clusters of Figure 7.4. (a) A packing returned by the ISA algorithm when applied to the Path49 instance of Wang and Valenzuela [151]. (b) A packing returned by the ISA algorithm when applied to the T6b instance of Hopper and Turton [92]. (c) A packing returned by the ISA algorithm when applied to the C6-P1 instance of Hopper and Turton [92]. (d) A packing returned by the ISA algorithm when applied to the Nice50 instance of Valenzuela and Wang[151]. The parameter n denotes the number of items in the instance.*

The last cluster is composed of instances containing items of a square shape that are fairly similar in size, as indicated by the range of values of the respective maximum aspect ratio and

the maximum area ratio (see Table 7.3 and Figure 7.5(a)–(b)). Furthermore, the value of the corresponding heterogeneity ratio is close to 1, which suggests that the instances belonging to this fourth cluster are heterogeneous. The cluster is also populated by instances that contain a relatively large number of wide items (the width ratio also lies within a range of large values). The nice set of Valenzuela and Wang [151], and the ngcut instances of Beasley [14] are among the instances in this cluster. An illustrative example of an instance in Cluster 4 is given in Figure 7.6(d). The packing in Figure 7.6(d) was generated by the ISA algorithm when applied to the Nice50 instance of Valenzuela and Wang [151].

Factors		Cluster 1	Cluster 2	Cluster 3	Cluster 4
	Min	1	0.0005	0.15	0.7
	Max	1	0.204	0.625	1
Heterogeneity	Mean	1	0.06	0.29	1
Ratio	Std	0	0.04	0.09	0.015
	Mode	1	0.05	0.3	1
		Strongly heterogeneous	**Significantly homogeneous**	**Significantly homogeneous**	**Heterogeneous**
	Min	1.56	0.22	0.95	0.23
	Max	43.3	200.9	38.3	68.01
Width	Mean	9.68	9.18	5.38	10.62
Ratio	Std	8.28	13.97	4.73	12.13
	Mode	8	5.37	1.78	4
		Predominantly narrow items	**Predominantly wide items**	**Predominantly narrow items**	**Predominantly wide items**
	Min	8.21	1.5	1	1.31
	Max	44 605	1 081 080	2 610	408
Maximum	Mean	529.48	6 593.81	162.76	98.07
Area Ratio	Std	3 266.37	69 992.9	361.17	33.6
	Mode	46.3	8.88	50	6.5
		Predominantly uneven sized items	**Predominantly uneven sized items**	**Predominantly equally sized items**	**Predominantly equally sized items**
	Min	51.14	1.5	1.44	1.625
	Max	100	322	41	48.98
Maximum	Mean	88.92	26.38	11.44	7.71
Aspect Ratio	Std	13.73	33.35	8.34	10.15
	Mode	80	3	10	5
		Predominantly rectangular items	**Predominantly rectangular items**	**Predominantly square items**	**Predominantly square items**
	Total	321	740	251	406

TABLE 7.3: *Clustering output according to the different factors described in §7.1. The rows labelled 'Min', 'Max', 'Mean', 'Std' and 'Mode' contain the minimum value, the maximum value, the mean value, the standard deviation and the mode of the different factors, respectively. The row labelled 'Total' contains the number of instances included in each cluster. The characteristics of each cluster are described in boldface.*

7.4 Chapter Summary

A cluster analysis was performed in this chapter in respect of the SPP benchmark data described in Chapter 5. The characteristics of the resulting clustered data were also presented and interpreted. The underlying features that best describe the benchmark instances were first discussed in §7.1. Thereafter, the clustering process followed and the evaluation procedure adopted were explained in some detail in §7.2. Finally, the characteristics of the clustered benchmark data were elucidated in §7.3.

Part III

New Strip Packing Algorithms

CHAPTER 8

Improved Strip Packing Metaheuristics

Contents

In this chapter, two adaptations are suggested to existing SPP algorithms. As demonstrated later in this dissertation, the first algorithm is an improvement on the IA algorithm of Wei *et al.* [158], while the second algorithm is a modification of the SPGAL algorithm of Bortfeldt [25]. A detailed description of the first algorithm is presented in §8.1. The algorithm consists of a hybrid approach in which the method of simulated annealing is combined with a heuristic construction algorithm. The working of the heuristic construction algorithm, as well as the overall procedure, is explained in the aforementioned section. The second improved algorithm involves application of the method of simulated annealing to solve an SPP problem instance directly in the space of the completely defined layout, without encoding of solutions. A detailed description of the working of the algorithm, together with a pseudocode representation of the overall procedure, is provided in §8.2. The chapter finally closes in §8.3 with a brief summary of its contents.

8.1 The Modified Improved Algorithm

According to the computational results returned by recently proposed algorithms, the IA algorithm of Wei *et al.* [158] is one of the best algorithms for solving the 2D SPP [158]. As described in §4.2.7, the IA algorithm is executed in three stages: Greedy selection, local improvement, and randomised improvement. The greedy selection stage involves generation of different orderings of the list of items to be packed, and the best solution thus obtained provides an initial solution for the second stage. The subsequent local improvement is aimed at improving the current solution. The procedure involves swapping pairs of items in a certain fixed order, accepting a swap that leads to an improved solution. During the third stage, a simple randomised algorithm is

implemented to further improve the solution. This stage involves swapping the packing order of two randomly selected items and accepting a swap only if it results in an improved solution.

The IA algorithm is based on the ISA approach of Leung *et al.* [116]. There are, however, three key differences between the two algorithms: First, the constructive heuristic embedded in the IA algorithm employs an eight-case scoring rule for selecting the best-fit item to pack, whereas the ISA algorithm employs a four-case scoring rule for this purpose. Secondly, a simple randomised improvement, which only accepts non-worsening solutions, is implemented in the IA algorithm with a view to find better solutions, whereas a simulated annealing technique is employed for this purpose in the ISA algorithm. Finally, a greedy selection procedure is implemented at the beginning of the IA algorithm to generate a good initial solution, whereas a multi-start strategy is embedded in the simulated annealing procedure in the case of the ISA algorithm, aimed at enhancing the search capability of the algorithm.

The use of the evaluation rule based on eight different cases for selecting appropriate items to pack is the principal advantage of the IA algorithm in terms of generating good results. While studying these results, however, the author found that some improvements could be made to the algorithm. The randomised algorithm employed in the IA algorithm is not a powerful random-based metaheuristic. Since the procedure only accepts non-worsening solutions, certain solution regions may not be visited during the search process, resulting in a small improvement or even a non-improvement of the solution. Moreover, a time limit of at most 60 seconds was originally imposed during the execution of the IA algorithm, causing the algorithm to terminate prematurely in some cases during the process of executing the local improvement for medium-sized and large instances. In such cases, the randomised algorithm has little or no effect on solutions.

A modified IA algorithm, referred to here as the *IAm algorithm*, is therefore proposed in this section in order to incorporate several essential improvements aimed at addressing the aforementioned disadvantages. The IAm algorithm consists of a hybrid approach in which the method of simulated annealing is combined with a heuristic construction algorithm. The constructive heuristic algorithm originally utilised in the IA algorithm is also adopted here, but with a slight modification. The local improvement stage embedded in the IA algorithm is omitted, and the randomised improvement procedure is replaced by the method of simulated annealing in the IAm algorithm. Finally, the greedy selection stage in the IA algorithm is included in the form of a multi-start strategy within the simulated annealing search process of the IAm algorithm.

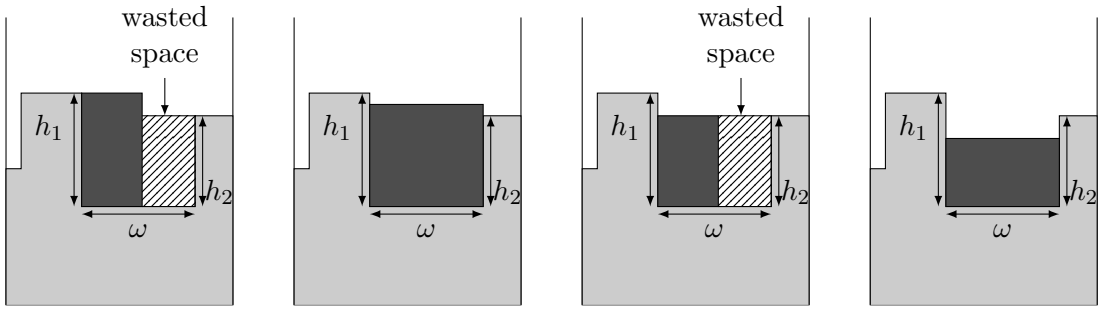
The newly proposed algorithm aims to leverage the ability of the random-based simulated annealing search to produce high-quality solutions within reasonable time frames. The non-inclusion of the original local improvement method in the IAm algorithm enables the simulated annealing procedure to improve on the solution quality of the incumbent effectively, thus largely avoiding the possibility of no improvement of solutions for large instances. Moreover, the multi-start strategy incorporated in the simulated annealing procedure enhances the search process with a view to explore unvisited portions of the solution space. Finally, the proposed heuristic construction algorithm generates denser packing layouts, thus reducing untidy arrangements that might occur during the early stage of the packing process.

8.1.1 Heuristic Construction Algorithm

Given a list of items to be packed, the heuristic originally embedded in the IA algorithm repeatedly selects the most suitable item according to a scoring rule and places the selected item at the lowest and left-most position in the strip. As described in §4.2.7, the scoring rule involves the evaluation of eight cases, giving priority to an item that fits perfectly into the available space.

The item with the largest score is selected for packing and the first item encountered in each case is considered for packing if several items exhibit the same maximal score.

The aforementioned scoring rule can, however, be modified so that the packing solution achieves a denser layout. According to the original scoring rule, a relatively tall and thin item is assigned a higher score than a relatively short and wide item (see Figure 8.1). This may sometimes lead to some waste, represented by the dashed area in Figure 8.1(a), which may not be filled if the minimum width of unpacked items is greater than the width of that space. In the modified scoring rule, this disadvantage is addressed by giving priority to an item that has the same width as the width of the available space. That is, cases (b) and (d) in Figure 8.1 achieve scores of 4 and 3, respectively, whereas cases (a) and (c) are assigned scores of 2 and 1, respectively. A summary of the modified scoring rule is provided in Table 8.1 for the case $h_1 \geq h_2$, where h_1 and h_2 represent the heights of left wall and the right wall of a selected space for packing, respectively. There are eight similar cases for $h_1 < h_2$. An illustrative example of the new scoring rule is shown in Figure 8.2.



(a) *Score = 4* (b) *Score = 3* (c) *Score = 2* (d) *Score = 1*

FIGURE 8.1: *An illustrative example of the scoring rule utilised by Wei* et al. *[158]. The parameters h_1, h_2, and ω are the height of the left wall, the height of the right wall, and the width of a selected available space, respectively. The light-shaded area represents items already packed, while the dark-shaded area is the newly packed item being scored. The dashed area represents wasted space (into which no unpacked item can fit).*

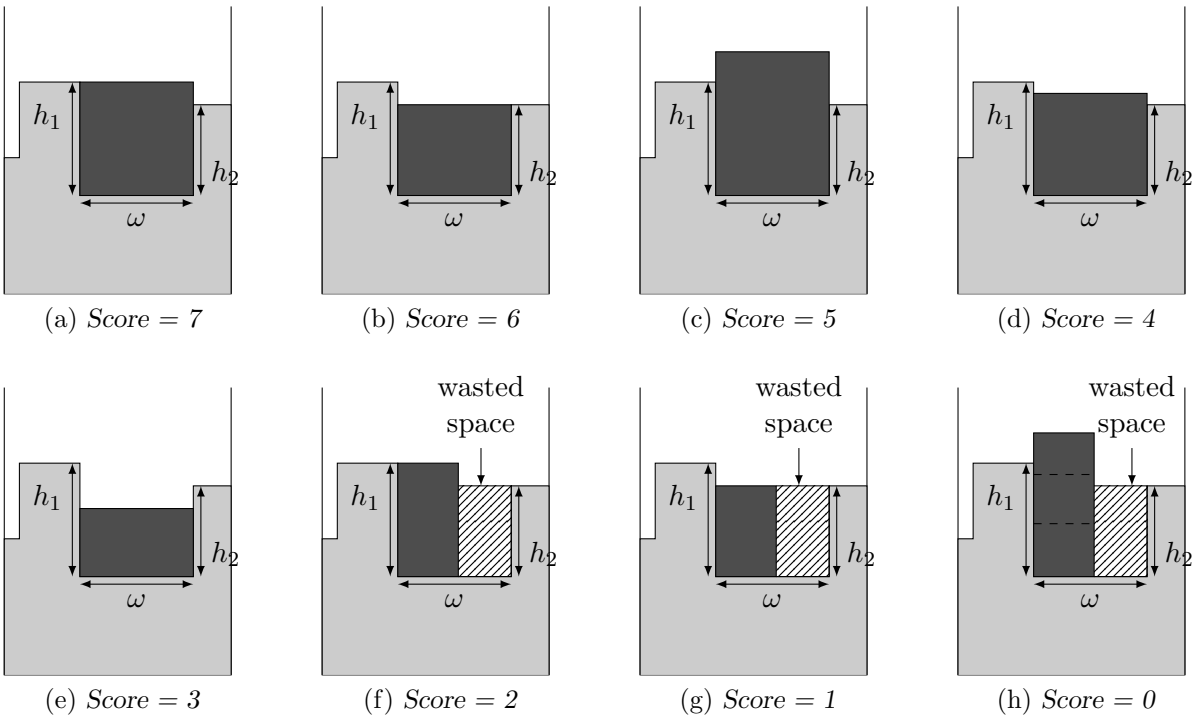
If	Conditions	Score
	ω = item.width and h_1 = item.height	7
	ω = item.width and h_2 = item.height	6
	ω = item.width and h_1 < item.height	5
$h_1 \geq h_2$	ω = item.width and h_1 > item.height	4
	ω = item.width and h_2 > item.height	3
	ω > item.width and h_1 = item.height	2
	ω > item.width and h_2 = item.height	1
	ω > item.width and $h_1 \neq$ item.height	0

TABLE 8.1: *The new scoring rule employed in the IAm algorithm for determining the best-suited item to pack into a selected available space. The parameters h_1, h_2, and ω denote the height of the left wall, the height of the right wall and the width of the available space, respectively.*

The heuristic construction algorithm here follows the same steps as in the CH algorithm proposed by Leung *et al.* [116], described in §3.5. The heuristic finds the lowest and left-most available space, and scores each unpacked item for that space according to the above rule. The item with the largest score value is selected and is packed in that space adjacent to the tallest neighbour.

If, during the packing process, the item selected for packing corresponds to one of the cases (f), (g), or (h) in Figure 8.2, an additional procedure is performed before placing it in the

corresponding space. This additional procedure involves attempting to reduce the resulting wasted area, represented by the dashed area in Figures 8.2(f)–(h). The procedure does so by selecting an unpacked item of largest width that fits into the space, packing it in the space being considered if its height is equal to the height of the incumbent item.



(a) *Score = 7* (b) *Score = 6* (c) *Score = 5* (d) *Score = 4*

(e) *Score = 3* (f) *Score = 2* (g) *Score = 1* (h) *Score = 0*

FIGURE 8.2: *Examples of the new scoring rule employed in the IAm algorithm. The parameters h_1, h_2, and ω denote the height of the left wall, the height of the right wall, and the width of a selected available space, respectively. The light-shaded area represents items already packed, while the dark-shaded area is the newly packed item being scored. The dashed area represents wasted space (into which no unpacked item can fit).*

8.1.2 The overall IAm Algorithm

The IAm algorithm initialises by generating a starting solution at random, and choosing an initial value for the temperature. Thereafter, it applies the heuristic construction algorithm described in §8.1.1 to the initial packing permutation and records the packing height returned as the best solution found so far. The algorithm further executes by repeatedly carrying out the following steps until a pre-determined termination condition is satisfied: First, a new neighbouring solution is generated by means of a suitable manipulation operator. Here, the manipulation operator consists of swapping the packing order of two randomly selected items from the incumbent solution. Thereafter, the heuristic construction algorithm is applied to the new packing permutation and the packing height returned is recorded as the current solution. Upon execution of this process, the current packing solution is compared with the best solution found so far. If it achieves a better packing layout than the best one recorded, it becomes the new best solution. Otherwise, it is rejected, unless a probability condition of accepting worsening solutions is satisfied.

During the execution of the above steps, the temperature is held constant. The value of this parameter is updated according to a pre-defined epoch management policy. Once an epoch

is terminated, the IAm algorithm executes a multi-start process. During this process, the currently generated permutation solution is sorted either according to non-increasing width or non-increasing perimeter or non-increasing height or non-increasing area. A pseudocode representation of the IAm algorithm is given in Algorithm 8.1.

Algorithm 8.1: **The IAm algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, the strip width W, and all the parameters required for implementation of the SA algorithm.

Output: A feasible packing of the items in $\mathcal{P}$ into a strip of width W.

```
generate an initial solution X_0, X ← X_0 ;
T ← T_0 the current value of the temperature;
BestSolution ← HeuristicConstruction(X);
while the stopping criterion is not yet satisfied do
    for a specific length of epoch do
        define a neighbourhood structure of the current solution X by applying a
        manipulation operator;
        obtain a new neighbouring solution X';
        CurrentSolution ← HeuristicConstruction(X');
        if CurrentSolution is better than BestSolution then
            BestSolution ← CurrentSolution;
        else
            if the probability condition of accepting a worsening solution is satisfied then
                BestSolution ← CurrentSolution;
    update the temperature T;
    randomly select one sorting strategy, either by decreasing perimeter or by decreasing
    height or by decreasing area or by decreasing width;
return BestSolution;
```

8.2 The SPSAL Algorithm

As mentioned in §2.2.3, three types of SPP solution approaches involving GAs are available in the literature. Approaches in the first category involve the use of hybrid techniques, whereby a GA is combined with some placement routine. The second type of GA solution methodology incorporates some of the layout information in the encoded solutions and employs an additional rule to generate the complete layout. Approaches in the third category attempt to solve the problem directly in the space of the fully defined layout, without any encoding of solutions. In applications of the SA technique within the context of the SPP in the literature, on the other hand, the majority of the proposed approaches involves the use of hybrid techniques, in which SA is combined with a heuristic decoding routine. To the best knowledge of the author, no SPP solution approaches involving SA in the second or third aforementioned categories have yet been proposed.

The SA packing technique, referred to as the *SPSAL algorithm*, proposed in this section is a solution approach in the third category mentioned above. It is an adaptation of the SPGAL algorithm of Bortfeldt [25], described in §4.2.3. This new solution approach is aimed at demon-

strating that the method of SA without any encoding of solutions is also suitable for application to the 2D SPP.

As described in §4.2.3, the SPGAL algorithm consists of solving multiple instances of a two-dimensional container loading problem iteratively until a smallest-length container (equivalent to a smallest packing height in the two-dimensional SPP) is reached. The CLP-GA genetic algorithm of Bortfeldt and Gehring [26] was implemented within this algorithm to solve each container loading problem instance. In the newly proposed SPSAL algorithm, multiple instances of a two-dimensional container loading problem are also solved iteratively until a smallest packing-height solution is obtained. The CLP-GA algorithm is, however, replaced by the method of SA, referred to as the *CLP-SA algorithm*, in the SPSAL algorithm to solve each container loading problem instance. Details of these algorithmic approaches are provided in this section.

8.2.1 The CLP-SA for the Container Loading Problem

Similar to the CLP-GA algorithm described in §4.2.3, the CLP-SA search procedure consists of solving a container loading problem instance directly in the space of a completely defined layout with a layer structure. A layout consists of successive rectangular layers in which one or more items are arranged. The representation of a layout, as a data structure for encoding solutions when applying the method of SA, is summarised in an array of quadruples (4-tuples), as described in §4.2.3. The information about the packing layout thus captured includes the total number of layers, the covered area, the layer records, and the item placings within a layer.

The overall procedure of the CLP-SA is presented in pseudocode form in Algorithm 8.2. Taking as inputs the container dimensions and the list of items to be packed, the SA search starts by initialising a feasible layout solution. During each subsequent iteration, a new solution is generated by applying a manipulation operator. During the course of this operation, the number ℓ of layers in the current solution is transferred to a new partial solution. The parameter ℓ is an integer chosen at random from the interval $[1, \ldots, \frac{L}{2}]$, where L is the total number of layers in the incumbent solution. Thereafter, this partial solution is completed by means of the FillingLayer() procedure described in Algorithm 4.4.

Upon execution of this process, the current packing solution is compared with the best solution found so far in terms of packing density (the covered part of the container area). If the incumbent solution yields a more dense packing layout than the previous best one, then it becomes the new best solution. Otherwise, it is rejected unless a probability condition of accepting worsening solutions is satisfied. Upon termination of an epoch, the temperature value is updated and the entire process is repeated until a pre-defined stopping criterion is satisfied.

8.2.2 The overall SPSAL Algorithm

The overall procedure of the SPSAL algorithm is given in pseudocode form in Algorithm 8.3. The algorithm takes as inputs the strip width and the dimensions of the items to be packed. At initialisation of the procedure, a starting solution is generated. Whereas the starting solution for the SPGAL algorithm is generated by invoking the BFDH* algorithm, the greedy selection procedure given in pseudocode form in Algorithm 8.4 is utilised for this purpose in the SPSAL algorithm. Given a list of items to be packed, the procedure first generates four different packing orders, which are obtained by sorting the list of items in non-increasing perimeter size, area, height, and width. Thereafter, the BFDH* algorithm is applied to each of the packing orders and the packing heights returned are saved. At the end of the procedure, the ordered list that

Algorithm 8.2: **CLP-SA algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, the dimensions H and W of the container, and all the parameters required for implementation of the SA algorithm.

Output: A feasible packing of the items in $\mathcal{P}$ into the container.

```
generate a feasible solution;
BestSolution ← InitialSolution;
while the stopping criterion is not yet satisfied do
    for a specific length of epoch do
        apply a manipulation operator to the current solution to generate a new partial solution;
        complete the new partial solution by means of the FillingLayer() procedure;
        CurrentSolution ← NewSolution;
        if CurrentSolution is better than BestSolution then
            BestSolution ← CurrentSolution;
        else
            if the probability condition of accepting a worsening solution is satisfied then
                BestSolution ← CurrentSolution;
    update the temperature;
return BestSolution;
```

yields the smallest packing height is returned and the corresponding solution is taken as the starting solution, denoted by *Sstart*.

Algorithm 8.3: **The SPSAL algorithm**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.

Output: A feasible packing of the items in $\mathcal{P}$ into a strip of width W.

```
generate a start solution Sstart using InitialSelection();
Sbest ← Sstart, Hbest ← h(Sstart);
if n > nplarge then
    determine the subset P' and keep the remaining layer of Sstart with the highest filling rates in Skept;
else
    P' ← P, Skept ← ∅;
initialise container length H ← Hbest − 1 − h(Skept);
while the obtained solution s does include all items in P' do
    solve CLP-SA for current length H, list of items P' and get a new solution s;
    Sbest ← s ∪ Skept, Hbest ← h(s) + h(Skept);
    H ← h(s) − 1;
post-optimise Sbest;
```

At this stage, the starting solution is recorded as the best solution found so far and the corresponding packing height is saved as the smallest container length achieved, denoted by *Hbest*. The next step of the SPSAL algorithm depends on the problem instance size. If the number of items in the problem instance considered exceeds a limit value, denoted by *nplarge*, then a subset

Algorithm 8.4: **Initial Selection algorithm** (InitialSelection())

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, and the strip width W.
Output: The best packing order with the smallest packing height.

$X \leftarrow$ an initial permutation;
Sort X by non-increasing perimeter to obtain X_p and $H_p \leftarrow$ BFDH*(X_p);
Sort X by non-increasing area to obtain X_a and $H_a \leftarrow$ BFDH*(X_a);
Sort X by non-increasing height to obtain X_h and $H_h \leftarrow$ BFDH*(X_h);
Sort X by non-increasing width to obtain X_w and $H_w \leftarrow$ BFDH*(X_w);
$BestH \leftarrow \min\{H_p, H_a, H_h, H_w\}$ and $X_B \leftarrow$ the packing order that leads to $BestH$;
return X_B and $BestH$;

$\mathcal{P}'$ is extracted from the initial set of items $\mathcal{P}$ for further processing. The subset $\mathcal{P}'$ is obtained by first examining all layers in *Sstart* in ascending order according to the filling rate. For each layer, the set $\mathcal{P}'$ is extended by all items placed in the layer. This process terminates when the number of items transferred in $\mathcal{P}'$ exceeds a predefined limit value, denoted by *npsmall*. The remaining layers of *Sstart* with the largest filling rates that were not transferred in $\mathcal{P}'$ are kept in a partial solution, denoted by *Skept*.

The next step of the SPSAL procedure consists of iteratively solving a container loading problem formed by the instance $\mathcal{P}'$ until no solution including all items in $\mathcal{P}'$ is found. At the beginning of this loop, the container length H is initialised by the difference $Hbest - 1 - h(Skept)$, where $h(Skept)$ is the length of the partial solution *Skept*. During each iteration, a container loading problem instance $\mathcal{P}'$ with container dimensions H and W is solved by invoking the CLP-SA algorithm. The packing solution returned is combined with the partial solution *Skept* to form a new solution of the corresponding SPP instance. The container length is reduced by one unit before the next iteration begins.

Upon execution of the above process, a post-optimisation heuristic is performed in respect of the packing solution returned. The post-optimisation procedure implemented in the SPGAL algorithm, described in §4.5, is adopted here in an attempt to fill unoccupied space between two successive layers. The heuristic consists of moving specific items from a particular layer to an adjacent layer so as to fill free space on the broad side of the adjacent layer. This procedure is repeated for several layers. Where the procedure is successful, an improved solution of the SPP is achieved.

8.3 Chapter Summary

Two adaptations of existing SPP metaheuristics were proposed and described in this chapter. The IAm algorithm was first presented in §8.1. The algorithm consists of hybrid SA solution approach in which a multi-start SA algorithm is combined with a heuristic construction algorithm. The SPSAL algorithm, described in §8.2, is an SA search technique which attempts to solve an SPP instance directly in the space of a fully defined layout, without any encoding of solutions.

CHAPTER 9

Parameter Fine-tuning of the Two Adapted Metaheuristics

Contents

The two adapted metaheuristics proposed in Chapter 8 are both SA packing techniques. In order to obtain the best results achievable by means of these algorithmic solution approaches, for the purposes of the comparative study carried out later in this dissertation, a suitable combination of the integrated SA algorithmic parameter values has to be selected for each case. This combination should ensure that the solution space is explored thoroughly during the SA search and that the two adapted algorithms return, on average, solutions of high quality for the SPP benchmark instances of Chapter 7.

A limited computational study based on an experimental design was conducted for this purpose. The experimental design consisted of testing various parameter settings of the SA technique embedded in each of the adapted algorithms. Details of the experimental study, as well as the results obtained, are reported and interpreted in this chapter.

The first section of this chapter is devoted to a brief description of the performance evaluation measures employed and the statistical analyses carried out. Thereafter, a presentation of the specific implementation of the SA algorithm employed in the two adapted metaheuristics follows in §9.2. Details of the experimental design followed are provided in §9.3, while the underlying

computational results are presented in §9.4. The chapter closes in §9.5 with a summary of the chapter contents.

9.1 Evaluation of Strip Packing Algorithms

Two types of methods prevail in the SPP literature for evaluating the effectiveness of packing algorithms [42, 91, 131]. The first approach consists of a theoretical analysis in which the solution returned by an algorithm is compared with a known optimal solution to a problem instance. Authors such as Coffman *et al.* [42] and Baker *et al.* [10, 11] have theoretically evaluated the quality of solutions produced by certain strip packing heuristics in this manner. The second approach involves testing the performance of algorithms in respect of a set of benchmark problem instances in terms of valid computational measures. An example of such measures is the relative difference between the packing height returned by an algorithm and the height of an optimal solution, often expressed in terms of a percentage gap or ratio. Hopper and Turton [91], Bortfeldt [25], and Leung *et al.* [116] have employed such a computational evaluation approach to test the relative performances of strip packing algorithms.

In the remainder of this dissertation, SPP algorithms are compared according to the latter approach in terms of two computational measures and are subjected to a statistical analysis in order to test their relative performances at a given level of statistical significance. More specifically, the various algorithms are compared in terms of both solution quality and computation time. Standard statistical analysis tests, such as ANalysis Of VAriance (ANOVA) and the Friedman test, are performed to compare the differences between the mean packing heights produced by these algorithms. The evaluation measures, as well as the statistical analysis tools employed in this dissertation, are described in the following sections.

9.1.1 Evaluation Measures

In order to evaluate strip packing algorithms with respect to solution quality, the so-called *strip packing accuracy* is employed. Given a strip packing algorithm $\mathtt{A}$ and a set of items $\mathcal{P}$ to be packed into a strip of pre-specified width, the packing accuracy α_A is defined as

$$\alpha_A = \frac{\mathtt{A}(\mathcal{P})}{\mathtt{OPT}(\mathcal{P})}, \tag{9.1}$$

where $\mathtt{A}(\mathcal{P})$ denotes the packing height achieved by the algorithm $\mathtt{A}$ for the packing set $\mathcal{P}$ and $\mathtt{OPT}(\mathcal{P})$ is the optimal packing height achievable for the set $\mathcal{P}$. This ratio reflects the relative percentage gap between the height of a packing solution achieved by the algorithm and that of an optimal packing solution. Since optimal solutions are not known for all benchmark instances considered in this dissertation, however, the optimal packing height is sometimes estimated by a valid lower bound [122]. In the remainder of this dissertation, the lower bound

$$LB = \frac{\sum_{i \in \mathcal{P}} h(i) \times w(i)}{W}$$

is used for this purpose, where W denotes the width of the strip, and $h(i)$ and $w(i)$ represent the height and width of item i in the packing set $\mathcal{P}$, respectively.

To compare the relative performances of the various algorithms with respect to computational time, the so-called *strip packing time efficiency* is employed. Given a strip packing algorithm $\mathtt{A}$ and a set of items $\mathcal{P}$ to be packed, the respective packing time efficiency $t^{\mathtt{A}}_{\mathcal{P}}$ is the time required

by the algorithm `A` to find a packing solution for the items in $\mathcal{P}$. This computation time may be measured by time tracking during the execution of the algorithm.

9.1.2 Statistical Analysis Tools

In order to compare the relative performances of the different SPP algorithmic approaches considered in this dissertation, a number of statistical significance tests from the realm of inferential statistics are carried out. These tests are employed to support a claim, also known as a *null hypothesis*, or to reject it in favour of a so-called *alternative hypothesis*, based on samples of data. The null hypothesis, denoted by H_0, is a statement of "no difference" between two observations and is assumed to be true *a priori*. The alternative hypothesis, denoted by H_1, on the other hand, is a statement of "difference" in this respect and usually corresponds to what a statistical hypothesis test is set up to establish. The rejection (or not) of H_0 is determined by a so-called *significance level*, denoted by α. The statistical test returns a so-called *p-value*, which represents the probability of obtaining a value that is at least as extreme as the observed value, given that H_0 is true. If the p-value is smaller than α, then H_0 is rejected at a significance level of α. The p-value may thus be interpreted as the smallest level of significance for which H_0 should be rejected.

Several statistical procedures are available in the literature, but it is generally suggested that *nonparametric tests* be employed in the context of metaheuristic comparisons, as these methods do not make any assumptions about the distribution of the underlying data [51]. In the remainder of this dissertation, the nonparametric *Friedman test* is employed in conjunction with the nonparametric *Nemenyi post hoc procedure* to test the difference between the mean packing heights returned by the different SPP algorithms. For interest's sake, a (parametric) ANOVA is also performed in conjunction with *Tukey's (parametric) post hoc test* to compare the performance of all algorithms.

An ANOVA may be utilised to test whether there is a significant difference between the means of samples that are normally distributed. The null hypothesis H_0 is that all the means of the samples are equal, while the alternative hypothesis H_1 is that the means are not all equal. Rejection of H_0 implies that there is a significant difference between at least two of the sample means. The procedure involves comparison of the ratio of between-group variance with within-group variance[1]. If the between-group variance is larger than the within-group variance, it indicates that the means are not equal. If the ANOVA reveals that there is a statistically significant difference between at least two of the sample means, a *post hoc* test is then required to determine between which pairs of sample means this difference actually occurs.

Tukey's HSD (Honest Significant Difference) *post hoc* test may follow an ANOVA test if the latter test's null hypothesis was rejected. It is a multiple-comparisons procedure aimed at locating differences between the means of pairs of samples. Tukey's HSD test achieves this by performing pairwise comparisons among the means of all samples based on the *studentised range distribution*[2]. It is suitable for multiple comparisons due to the

[1] An ANOVA involves use of the F-test and F-distribution to test the equality of means statistically. A variance ratio F is calculated by dividing the mean square within groups by the mean square between groups. The critical value in the F-distribution at a chosen significance level, denoted here by F_c, is then compared with the calculated value F. If $F > F_c$, then there is a statistically significant difference between at least two of the sample means.

[2] The studentised range distribution is a probability distribution that arises when estimating the range of a normally distributed population with an unknown standard deviation.

manner in which it limits the *Type I error* risk — the risk of an incorrect rejection of a true null hypothesis at a significance level of α [86].

The Friedman test is a non-parametric equivalent of an ANOVA. It ranks the algorithmic results for each benchmark instance and uses these ranks to test for significant differences, under a null hypothesis stating that all the algorithms return equivalent results in which case their average ranks should be equal[3]. If the Friedman test yields a positive result, *i.e.* the null hypothesis is rejected, then it can be concluded that at least two of the algorithms yield results that are significantly different from each other.

The Nemenyi *post hoc* test may be employed to compare the relative performances of algorithms with a view to identifying specific pairs of algorithms that perform differently. More specifically, the Nemenyi test performs pairwise significance tests between all pairs of samples using the Friedman ranks. The performances of two algorithms are significantly different if the corresponding mean ranks differ by at least a well-defined critical difference[4]. The Nemenyi test may be used to reduce the chances of obtaining false-positive results or *Type I errors* when performing multiple pairwise tests under the null hypothesis.

9.2 Simulated Annealing Implementation

This section contains a description of the particular implementation of the SA algorithm employed in both the IAm and SPSAL algorithms to obtain high-quality solutions with respect to the clustered SPP benchmark instances of Chapter 7. A discussion is included on the method of determination of an initial temperature for the algorithm as well as an initial solution, the cooling schedule employed, the epoch management policy implemented, and the termination criterion selected.

9.2.1 The Initial Solution

An initial solution is required at the beginning of the SA search for each of the two adapted algorithms. The implementation of the SA process employed in the IAm algorithm randomly generates an initial solution. This is achieved by generating a random feasible permutation of the items to be packed. In the case of the SPSAL algorithm, the method described in pseudocode form in Algorithm 8.4 is employed.

[3]The Friedman test statistic

$$F = \frac{12b}{k(k+1)} \sum_{i=1}^{k} \left(\bar{r}_i - \left(\frac{k+1}{2} \right) \right)^2$$

follows a Chi-square distribution with $k-1$ degrees of freedom, where $\bar{r}_i$ is the mean rank associated with sample i, k is the number of columns (samples), and b is the number of rows (observations).

[4]The Nemenyi procedure tests for significance by means of a critical distance

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

between the Friedman ranks, where k is the number of columns (algorithms in this case) and N is the number of rows (benchmark instances in this case). The critical value q_α is based on the Studentised range statistic for infinite degrees of freedom, divided by $\sqrt{2}$ [127].

9.2.2 The Initial Temperature

In order to calculate an appropriate value for the initial temperature, the method proposed by Van Laarhoven [152] is employed. As suggested by Van Laarhoven, the value of the initial temperature T_0 may be determined in such a way that it results in an average increase of the objective function within a reasonable acceptance probability. Such a value can be estimated by conducting an initial or trial search in which all deteriorating moves are accepted and in which the average objective function increase is observed. According to this approach, the initial temperature is given by

$$T_0 = -\frac{\overline{\Delta f}}{\ln(p_0)}, \tag{9.2}$$

where $\overline{\Delta f}$ is the average deterioration in objective function values and p_0 is the acceptance probability, defined as the number of accepted solutions that exhibit a deterioration in the objective function value during the initial search.

9.2.3 The Cooling Schedule

The geometric cooling schedule is implemented within the integrated SA algorithmic implementation of the two adapted SPP metaheuristics. This cooling schedule is the most popular in the literature and has been proven to be successful in solving SPP instances [28, 31, 52, 116]. The value of the parameter β embedded in this schedule is empirically determined in the next section.

9.2.4 The Epoch Management Policy

Two different epoch management rules are considered in this study. In the first case, the temperature is held constant for a fixed number of iterations, while an epoch is terminated once a certain number of successful moves have been attempted in the second case. These epoch management protocols have previously been utilised by Hopper and Turton [91], and also by Leung *et al.* [116] in their experimentations.

9.2.5 The Termination Criterion

The stopping criterion for the SA algorithm adopted in both IAm and SPSAL metaheuristics is based on the number of iterations executed. An SA run is terminated when 5 000 iterations have been executed. A time limit of 60 seconds is additionally imposed for large instances that contain more than 5 000 items.

9.3 Experimental Design

In order to identify the best results achievable by the two adapted SPP metaheuristics, a suitable combination of the embedded SA algorithmic parameter values has to be selected. The method of SA described above requires specification of three parameter values. These parameters are the acceptance probability p_0 for determining the initial temperature, the cooling parameter β, and the length of an epoch during the search.

Two values of each of these parameters were considered during the SA parameter optimisation experimental study. Busetti [32] claimed that a suitable value for p_0 is 0.8. This corresponds to a reasonable intial temperature that results in an average increase of the objective function with an acceptance probability of about 0.8. This value was adopted in the current experimental study, resulting in a value of T_0 equal to 45.1 in the SPSAL algorithmic implementation, while T_0 is equal to 67.2 in the case of IAm algorithmic implementation, when applying (9.2). A smaller value of p_0, equal to 0.5, was also chosen to assess the resulting relative performances of these algorithms. The corresponding value of T_0 are 14.5 and 21.6 for the SPSAL and IAm algorithmic implementations, respectively, according to the same formula.

In order to determine the value of the average deterioration in the objective function in (9.2), an initial iterative search procedure was conducted. Given an initial solution, a neighbouring solution was generated during each iteration of the search. This was achieved by means of a swap rule (reversing the packing order of any two randomly selected items in the solution) in the IAm algorithmic implementation, while the manipulation operator described in §8.2.1 was employed in the SPSAL algorithmic implementation. All increases in respect of the packing solution height were accepted and the average increase observed was finally recorded. A pseudocode listing of the procedure is provided in Algorithm 9.1.

Algorithm 9.1: **Determining the average deterioration $\overline{\Delta f}$ in (9.2)**

Input : A list $\mathcal{P}$ of items to be packed, the dimensions $\langle w(\mathcal{P}_i), h(\mathcal{P}_i)\rangle$ of the items, the strip width W, and the required number of iterations *MaxIter*.
Output: The average increase in the packing height.

```
counter ← 0;
Increase ← ∅;
BestH ← InitialPackingSolution;
for counter ← 0 to MaxIter do
    apply the manipulation operator to the current solution;
    obtain a new neighbouring solution CurrentH;
    CurrentH − BestH ← IncreasedH;
    if IncreasedH > 0 then
        add IncreasedH to the list Increase;
        counter ← counter + 1;
return mean(Increase);
```

Two values, namely 0.93 and 0.95, were considered for the cooling parameter. These values were chosen based on the results of an experimental study conducted by Leung *et al.* [116] in 2011. They studied the effect of varying the value of β, by considering six different values of the parameter in their proposed packing algorithms. They found that for values of β equal to 0.93 and 0.95, their packing algorithms yielded the best and the second best packing results, respectively.

With regard to the epoch management policies, two different methods were implemented. In the first case, the temperature was held constant for a fixed number N of iterations, where N denotes the problem instance dimension. In the second case, an epoch was terminated when a total number $\frac{N}{2}$ of successful moves had been attempted during the algorithmic search.

Eight implementations were subsequently compared with one another in respect of the clustered benchmark instances of Chapter 7 for each of the two improved algorithms. Tables 9.1–9.2 provide summaries of these algorithmic incarnations together with the corresponding parameter

values. These implementations were each executed five times, and the best solution returned was recorded in each case.

All algorithms were coded in Python using Spyder Version 2.7.6. The algorithms were furthermore all executed on an Intel Core i7-4790 CPU running at 3.60 GHz with 8 GB RAM in the Ubuntu 14.04 operating system.

	$p_0 = 0.8$ ($T_0 = 67.2$)		$p_0 = 0.5$ ($T_0 = 21.6$)	
	$\beta = 0.93$	$\beta = 0.95$	$\beta = 0.93$	$\beta = 0.95$
Epoch length: N moves	*IAm1*	*IAm3*	*IAm5*	*IAm7*
Epoch length: $\frac{N}{2}$ successful moves	*IAm2*	*IAm4*	*IAm6*	*IAm8*

TABLE 9.1: *Eight different incarnations of the IAm algorithm, based on different values of the algorithmic parameters, considered during the parameter evaluation experiment. The parameter p_0 represents the acceptance probability for calculating the initial temperature T_0, while the other two parameters, β and N, denote the cooling parameter and the problem instance dimension (the number of items involved in a given instance), respectively. The table entries are the names assigned to the experiments.*

	$p_0 = 0.8$ ($T_0 = 45.1$)		$p_0 = 0.5$ ($T_0 = 14.5$)	
	$\beta = 0.93$	$\beta = 0.95$	$\beta = 0.93$	$\beta = 0.95$
Epoch length: N moves	*SPSAL1*	*SPSAL3*	*SPSAL5*	*SPSAL7*
Epoch length: $\frac{N}{2}$ successful moves	*SPSAL2*	*SPSAL4*	*SPSAL6*	*SPSAL8*

TABLE 9.2: *Eight different incarnations of the SPSAL algorithm, based on different values of the algorithmic parameters, considered during the parameter evaluation experiment. The parameter p_0 represents the acceptance probability for calculating the initial temperature T_0, while the other two parameters, β and N, denote the cooling parameter and the problem instance dimension (the number of items involved in a given instance), respectively. The table entries are the names assigned to the experiments.*

9.4 Computational Results

The numerical results obtained when following the experimental design described in §9.3 are presented in this section. First, the results returned by the eight IAm implementations are reported in §9.4.1. Thereafter, the results returned by the eight SPSAL implementations are discussed in §9.4.2. The results are reported in the form of boxplots and a significance level of $\alpha = 0.05$ is adopted for all the results presented.

9.4.1 Results obtained by the IAm Algorithm

Boxplots of the results obtained by the eight IAm implementations described in Table 9.1, when applied to the clustered benchmark data of Chapter 7, are shown in Figure 9.1. A Friedman test performed on the results yielded p-values less than 1×10^{-15} for Clusters 1, 2, and 4, and a p-value 0.0062 for Cluster 3, suggesting that there are statistically significant differences between the solutions achieved by some of the implementations at a 5% level of significance.

Performing the Nemenyi test in respect of the eight sets of solutions for the first benchmark cluster indicated that there is no significant difference between the results returned by the

IAm2 and IAm6 implementations, while their results differ statistically from those of the other implementations at a 5% level of significance. The mean performance ratio achieved by these two implementations are better than those achieved by the other six implementations, as shown in Table 9.3. The same observations apply to the eight sets of solutions returned in respect of the second and fourth benchmark cluster (see Tables 9.4–9.6). Accordingly, the IAm2 and IAm6 are statistically similar in performance for the first, second and fourth benchmark clusters. The IAm6 implementation is selected arbitrarily as the preferred IAm algorithmic implementation in respect of Clusters 1 and 4 data, while the IAm2 implementation is selected arbitrarily as the preferred IAm algorithmic implementation in respect of Cluster 2 data in the comparative study carried out later in this dissertation.

Performing the Nemenyi test in respect of the eight sets of solutions for the third benchmark cluster indicated that there is no statistical difference at a 5% level of significance between the results returned by any pair of implementations, which is clear from the p-values of this test shown in Table 9.5. Applying an ANOVA to the results yielded a p-value of 1 and Tukey's HSD *post hoc* test performed on the solutions suggested that the mean ranks achieved by all pairs of implementations are statistically indistinguishable at a 5% level of significance. These outputs may indicate that the Friedman test returned false-positive results and that the null hypothesis (that all the implementations return equivalent results with equal mean ranks) may not be rejected for this cluster. These results were verified and validated by Kidd [108], a statistician at the Centre for Statistic Consultation at Stellenbosch University.

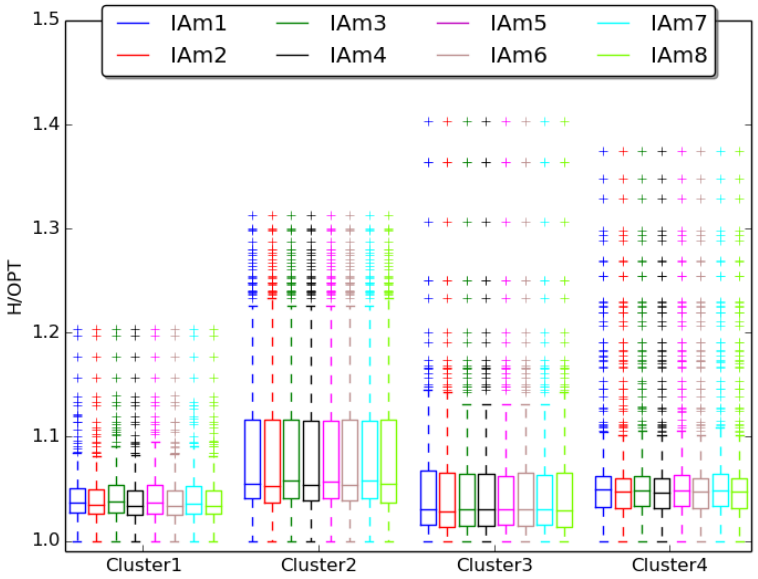


FIGURE 9.1: *The distribution of results returned by the eight IAm implementations in Table 9.1 when applied to the clustered benchmark data of Chapter 7. The ratio H/OPT represents the packing accuracy in (9.2), which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT.*

According to the statistical analysis result above, the eight IAm implementations performed statistically similarly for the third benchmark cluster at a 5% of statistical significance. The IAm2 implementation is selected arbitrarily as the preferred IAm algorithmic implementation in the comparative study carried out later in this dissertation.

In summary, the performance of the IAm algorithm is sensitive with respect to the cooling parameter value and the epoch length for the Cluster 1, 2 and 4 data. A cooling parameter value of 0.93 is statistically suggested for adoption in the IAm algorithm in respect of these

p-values of the Nemenyi rank test: IAm algorithm (Cluster 1)								
Algorithm	IAm1	IAm2	IAm3	IAm4	IAm5	IAm6	IAm7	IAm8
IAm2	0.0073	—						
IAm3	0.816	9.4×10^{-6}	—					
IAm4	0.816	9.4×10^{-6}	1.0	—				
IAm5	1.0	0.0078	0.805	0.805	—			
IAm6	1.8×10^{-5}	0.877	3×10^{-9}	3×10^{-9}	2×10^{-5}	—		
IAm7	1.0	0.0036	0.901	0.901	1.0	6.9×10^{-6}	—	
IAm8	1.0	0.0036	0.901	0.901	1.0	6.9×10^{-6}	1.00	—
Mean (Rank)	1.0421 (2)	1.0403 (1)	1.0433 (2)	1.0433 (2)	1.0423 (2)	1.04 (1)	1.0425 (2)	1.0425 (2)

TABLE 9.3: *Comparison of the results returned by the eight IAm implementations in Table 9.1 in respect of the first cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

p-values of the Nemenyi rank test: IAm algorithm (Cluster 2)								
Algorithm	IAm1	IAm2	IAm3	IAm4	IAm5	IAm6	IAm7	IAm8
IAm2	5.7×10^{-3}	—						
IAm3	0.999	6×10^{-5}	—					
IAm4	0.999	6×10^{-5}	1.0	—				
IAm5	1.0	1.8×10^{-3}	1.0	1.0	—			
IAm6	0.035	0.956	0.0067	0.0067	0.015	—		
IAm7	0.985	7.1×10^{-6}	0.999	0.999	0.998	0.0013	—	
IAm8	0.985	7.1×10^{-6}	0.999	0.999	0.998	0.0013	1.00	—
Mean (Rank)	1.0748 (2)	1.073 (1)	1.0757 (2)	1.0757 (2)	1.075 (2)	1.0737 (1)	1.0754 (2)	1.0754 (2)

TABLE 9.4: *Comparison of the results returned by the eight IAm implementations in Table 9.1 in respect of the second cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

p-values of the Nemenyi rank test: IAm algorithm (Cluster 3)								
Algorithm	IAm1	IAm2	IAm3	IAm4	IAm5	IAm6	IAm7	IAm8
IAm2	0.59	—						
IAm3	0.97	0.99	—					
IAm4	0.97	0.99	1.0	—				
IAm5	1.0	0.92	1.0	1.0	—			
IAm6	0.83	1.0	1.0	1.0	0.99	—		
IAm7	0.99	0.97	1.0	1.0	1.0	1.0	—	
IAm8	0.99	0.97	1.0	1.0	1.0	1.0	1.00	—
Mean (Rank)	1.1015 (1)	1.0997 (1)	1.101 (1)	1.101 (1)	1.1007 (1)	1.101 (1)	1.1007 (1)	1.1007 (1)

TABLE 9.5: *Comparison of the results returned by the eight IAm implementations in Table 9.1 in respect of the third cluster of benchmark instances. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

p-values of the Nemenyi rank test: IAm algorithm (Cluster 4)								
Algorithm	IAm1	IAm2	IAm3	IAm4	IAm5	IAm6	IAm7	IAm8
IAm2	0.0076	—						
IAm3	1.0	0.0063	—					
IAm4	1.0	0.0063	1.0	—				
IAm5	1.0	0.0056	1.0	1.0	—			
IAm6	0.002	1.0	0.0016	0.0016	0.0014	—		
IAm7	0.949	6×10^{-5}	0.961	0.961	0.966	1.0×10^{-5}	—	
IAm8	0.949	6×10^{-5}	0.961	0.961	0.966	1.0×10^{-5}	1.00	—
Mean (Rank)	1.066 (2)	1.0645 (1)	1.0659 (2)	1.0659 (2)	1.066 (2)	1.0644 (1)	1.0661 (2)	1.0661 (2)

TABLE 9.6: *Comparison of the results returned by the eight IAm implementations in Table 9.1 in respect of the fourth cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

benchmark clusters, and epochs should be terminated when $\frac{N}{2}$ successful moves have been attempted during the algorithmic search for these three data clusters.

9.4.2 Results obtained by the SPSAL Algorithm

Boxplots of the results obtained by the eight SPSAL implementations described in Table 9.2, when applied to the clustered benchmark data of Chapter 7, are shown in Figure 9.2. Applying the Friedman test to the results obtained for all the clusters yielded p-values of 0.9 for Clusters 1, 3, and 4, and a p-value of 6.75×10^{-6} for Cluster 2, suggesting that the null hypothesis (that all the implementations return equivalent results with equal mean ranks) may only be rejected for the Cluster 2 data.

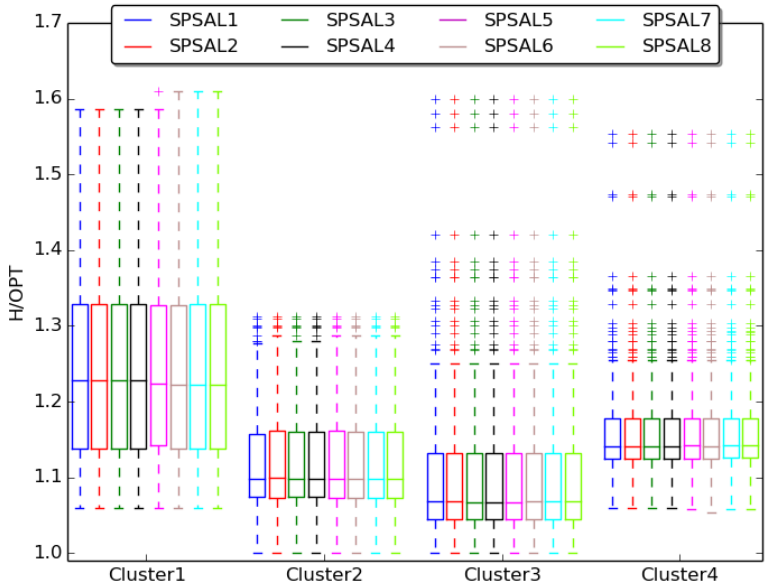


FIGURE 9.2: *The distribution of results returned by the eight SPSAL implementations in Table 9.2 when applied to the clustered benchmark data of Chapter 7. The ratio H/OPT represents the packing accuracy in (9.2), which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT.*

p-values of the Nemenyi rank test: SPSAL algorithm (Cluster 2)								
Algorithm	SPSAL1	SPSAL2	SPSAL3	SPSAL4	SPSAL5	SPSAL6	SPSAL7	SPSAL8
SPSAL1	—							
SPSAL2	1.0	—						
SPSAL3	1.0	1.0	—					
SPSAL4	1.0	1.0	1.0	—				
SPSAL5	0.12	0.34	0.21	0.21	—			
SPSAL6	0.19	0.47	0.32	0.32	1.0	—		
SPSAL7	0.34	0.67	0.5	0.5	1.0	1.0	—	
SPSAL8	0.34	0.66	0.5	0.5	0.99	1.0	1.0	—
Mean (Rank)	1.1042 (1)	1.1044 (1)	1.1041 (1)	1.104 (1)	1.1037 (1)	1.1035 (1)	1.1035 (1)	1.1035 (1)

TABLE 9.7: *Comparison of the results returned by the eight SPSAL implementations in Table 9.2 in respect of the second cluster of benchmark instances. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

The Nemenyi *post hoc* test was performed with respect to the eight results for the second cluster data in order to determine between which pairs of implementation outputs statistical differences are discernible. The output of the test, for which the p-values are provided in Table 9.7, indicates that there is no statistical difference at a 5% level of significance between the results returned by any pair of implementations for this cluster. Performing an ANOVA to the results yielded a p-value of 1 and the Tukey's HSD *post hoc* test applied to the results indicated that no pairwise mean differences were statistically evident at a 5% level of significance. These outputs may indicate that the Friedman test returned false-positive results and that the null hypothesis (that all the implementations return equivalent results with equal mean ranks) may not be rejected for this cluster. Again these results were verified and validated by Kidd [108].

According to this statistical analysis, the eight implementations performed relatively similarly for all four benchmark clusters. The post-optimisation heuristic performed at the end of the SPSAL procedure may be the reason for this relative performance invariance. Different packing layouts, with different packing height solutions, may be obtained from the eight implementations before the execution of the post-optimisation process, but these solutions may converge to a near-optimal solution after execution of the latter process. The SPSAL6 implementation is selected arbitrarily in further comparisons as SPSAL implementation for all four SPP benchmark clusters in this dissertation.

9.5 Chapter Summary

This chapter contained descriptions of the computational study conducted with respect to the identification of superior implementations of the two adapted algorithms of Chapter 8. The chapter opened in §9.1 with a presentation of the performance measures utilised and the statistical analysis tools implemented. This was followed by a description of the specific implementation of the method of SA employed in the two adapted algorithms in §9.2. Thereafter, details of the experimental design followed in §9.3. Finally, the computational results returned by both metaheuristics, when following the experimental design described in the previous section, were reported in §9.4.

A summary of the most effective algorithmic implementations for both the IAm and SPSAL algorithms may be found in Table 9.8 for each SPP benchmark data cluster. This table indicates

the appropriate parameter values to be selected for the algorithmic implementations of these metaheuristics.

		Cluster 1	Cluster 2	Cluster 3	Cluster 4
IAm	Initial temperature	21.6	67.2	67.2	21.6
	Cooling parameter	0.93	0.93	0.93	0.93
	Epoch length	Fixed number $\frac{N}{2}$ of successful moves	Fixed number $\frac{N}{2}$ of successful moves	Fixed number $\frac{N}{2}$ of successful moves	Fixed number $\frac{N}{2}$ of successful moves
SPSAL	Initial temperature	14.5	14.5	14.5	14.5
	Cooling parameter	0.93	0.93	0.93	0.93
	Epoch length	Fixed number $\frac{N}{2}$ of successful moves	Fixed number $\frac{N}{2}$ of successful moves	Fixed number $\frac{N}{2}$ of successful moves	Fixed number $\frac{N}{2}$ of successful moves

TABLE 9.8: *Summary of the recommended implementations for both the IAm and SPSAL algorithms with respect to each cluster of benchmark instances. The parameter N denotes the problem instance dimension.*

Part IV

The Relative Effectiveness of Different SPP Algorithmic Approaches

CHAPTER 10

Appraisal of Strip Packing Heuristics

Contents

In this chapter, the results obtained by the five strip packing heuristics described in Chapter 3, when applied to the clustered benchmark data of Chapter 7, are presented. First, a presentation and interpretation of the results returned by each of the heuristics are provided in §10.1. Thereafter, the comparative results of the five heuristics, in terms of both solution quality and execution time, are discussed in §10.2. All the results are represented in the form of boxplots together with the appropriate statistical analysis, as described in §9.1. Finally, a summary of the contents of the chapter, including a characterisation of the effectiveness of the heuristics in respect of the four benchmark data clusters, is provided in §10.3.

10.1 Results obtained by each Strip Packing Heuristic

The five strip packing heuristics described in Chapter 3 pack the list of items according to a given order either in a more explicit way or in an indirect manner. Certain algorithms, such as the BFDH* and BL algorithms, pack items one by one according to the order in which they have been sorted, while others, such as the BF and CH algorithms, dynamically search the list for the best-suited item to pack, resolving ties using the ordered list. Some researchers' results have shown that the initial packing ordering has a crucial effect on the performance of algorithms [29, 91, 115, 116]. It was thus decided to investigate the effect of different sorting strategies in the solution achieved by each heuristic in respect of the clustered benchmark instances of Chapter 7.

Four different sorting strategies are considered in the computational study, namely sorting the lists of items according to *non-increasing area*, according to *non-increasing height*, according to *non-increasing perimeter*, and according to *non-increasing width*. Each of the five algorithms

reviewed in Chapter 3 was implemented using these sorting strategies and their relative performances are compared in respect of the clustered benchmark instances of Chapter 7. The results obtained are reported in this section.

10.1.1 Results obtained by the BFDH* Algorithm

As described in §3.1, the BFDH* algorithm packs a list of items into consecutive strip levels by filling a level from left to right. It packs each item into the level in which it fits and achieves the minimum residual horizontal space. The aforementioned four sorting strategies were each applied to the BFDH* algorithm to sort the list of items to be packed prior to the packing procedure. Boxplots of the results returned by the four algorithmic implementations, when applied to the clustered benchmark instances of Chapter 7, are shown in Figure 10.1.

For each cluster, a similar trend is observed with respect to the order of the comparative performance of the four algorithmic implementations. The implementation employing the decreasing height sorting strategy performed the best overall, achieving the smallest mean packing height for each of the four clusters. The implementation employing the decreasing perimeter and the decreasing area sorting strategies performed respectively the second and the third most effectively overall. The implementation employing the decreasing width sorting strategy was consistently the worst performing implementation in respect of each cluster. The presence of significant differences between the results returned by these four implementations is elucidated in the Nemenyi rank test results in Tables 10.1–10.4.

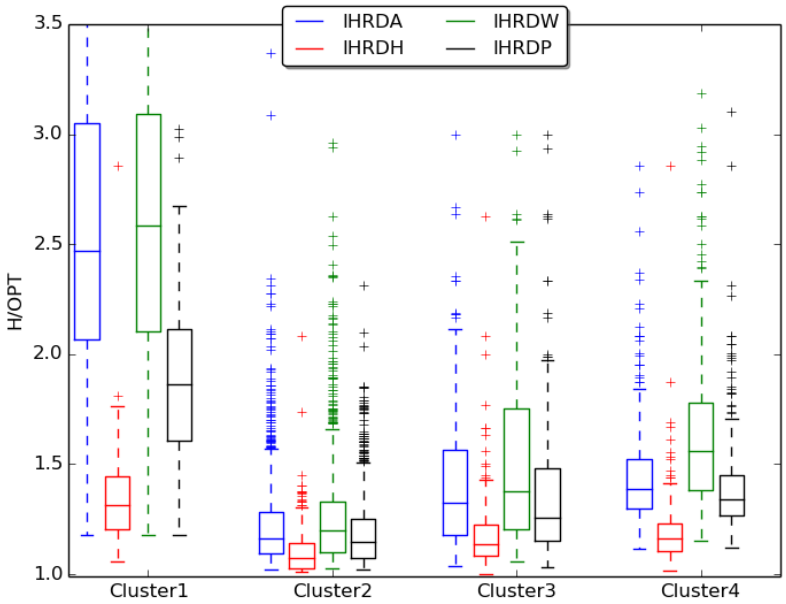


FIGURE 10.1: *The distribution of results returned during a preliminary study involving implementations of the BFDH* algorithm of §3.1 with four different sorting strategies described above, when applied to the clustered benchmark data. The ratio H/OPT represents the packing accuracy, which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT. The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

The level restriction prevailing in the BFDH* algorithm is the principal cause for its relatively good performance when implemented in conjunction with the decreasing height sorting strategy, as well as for its comparatively poor performance in conjunction with the other three sorting

p-values of the Nemenyi rank test: BFDH* algorithm (Cluster 1)				
Algorithm	BFDH*DA	BFDH*DH	BFDH*DP	BFDH*DW
BFDH*DH	$< 1 \times 10^{-15}$	—		
BFDH*DP	3.3×10^{-14}	3.5×10^{-14}	—	
BFDH*DW	0.46×10^{-2}	$< 1 \times 10^{-15}$	3.3×10^{-14}	—
Mean (Rank)	2.54 (3)	1.34 (1)	1.87 (2)	2.68 (4)

TABLE 10.1: *Comparison of four sorting strategies when applied to the BFDH* algorithm of §3.1 in respect of the first cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: BFDH* algorithm (Cluster 2)				
Algorithm	BFDH*DA	BFDH*DH	BFDH*DP	BFDH*DW
BFDH*DH	$< 1 \times 10^{-15}$	—		
BFDH*DP	5.9×10^{-9}	$< 1 \times 10^{-15}$	—	
BFDH*DW	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.24 (3)	1.10 (1)	1.21 (2)	1.29 (4)

TABLE 10.2: *Comparison of four sorting strategies when applied to the BFDH* algorithm of §3.1 in respect of the second cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: BFDH* algorithm (Cluster 3)				
Algorithm	BFDH*DA	BFDH*DH	BFDH*DP	BFDH*DW
BFDH*DH	3×10^{-14}	—		
BFDH*DP	1.1×10^{-3}	2.5×10^{-8}	—	
BFDH*DW	1.1×10^{-3}	$< 1 \times 10^{-15}$	5.2×10^{-13}	—
Mean (Rank)	1.49 (3)	1.21 (1)	1.43 (2)	1.63 (4)

TABLE 10.3: *Comparison of four sorting strategies when applied to the BFDH* algorithm of §3.1 in respect of the third cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: BFDH* algorithm (Cluster 4)				
Algorithm	BFDH*DA	BFDH*DH	BFDH*DP	BFDH*DW
BFDH*DH	$< 1 \times 10^{-15}$	—		
BFDH*DP	2×10^{-5}	$< 1 \times 10^{-15}$	—	
BFDH*DW	2.3×10^{-11}	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.45 (3)	1.18 (1)	1.39 (2)	1.65 (4)

TABLE 10.4: *Comparison of four sorting strategies when applied to the BFDH* algorithm of §3.1 in respect of the fourth cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

strategies. By implementing the non-increasing height sorting strategy in the BFDH* algorithm, it certainly returns a more dense packing layout as all tall items are packed at the start of the procedure and any existing gaps between levels may be filled with short unpacked items toward the end of the process. By employing a sorting strategy other than the non-increasing height strategy, on the other hand, tidy arrangements (*i.e.* with relatively small empty areas) may be obtained during earlier stages of the BFDH* solution but taller items are only packed at the end of the process, which would most likely contribute negatively to the overall packing height.

These performance differences are especially noticeable with respect to the first cluster of benchmark data (see Table 10.1). Since this cluster is mainly populated by instances containing long and flat, or tall and thin, rectangular items, the use of non-increasing height as a sorting strategy in the BFDH* algorithm is recommended in this case so as to avoid tall unpacked items at a later stage of the procedure contributing negatively to the solution's overall packing height. For a similar reason, the use the non-increasing height sorting strategy is also preferable for the implementation of the BFDH* algorithm with respect to the remaining three clusters.

10.1.2 Results obtained by the BL Algorithm

Boxplots of the results returned by the four BL algorithmic implementations, when applied to the clustered benchmark data of Chapter 7, are shown in Figure 10.2. It is clear from these boxplots that the implementation employing the non-increasing height sorting strategy achieved the smallest mean packing height for all clusters, outperforming the other three algorithmic implementations. These three implementations performed interchangeably the second best and the worst, depending on the benchmark cluster. This observation is supported by applying the Friedman test to the packing results for the various clusters, each yielding a p-value less than 1×10^{-15}, and by the Nemenyi test for which the p-values are provided in Tables 10.5–10.8.

As described in §3.2, the BL algorithm employs a sliding method to pack a given list of items. The process of packing an item starts by placing it in the top-right corner of the strip and then making successive moves by repeatedly sliding the item as far as possible to the bottom and to the left. This sliding technique mainly explains the relative superior performance of the BL algorithm when implemented in conjunction with the non-increasing height sorting strategy compared to the other three algorithmic implementations.

If implemented in conjunction with the non-increasing height sorting strategy, the BL algorithm packs tall items at the beginning of the procedure, resulting in a situation where any unoccu-

pied area between two tall items may be filled by short unpacked items. This results in denser packing layouts with relatively small packing heights. By employing the other sorting strategies in conjunction with the BL algorithmic implementation, on the contrary, dense layouts may be obtained, but with large unused spaces, resulting in taller items contributing to larger packing heights. According to these results, the use of non-increasing height sorting strategy is statistically suggested for incorporation in the BL algorithmic implementation for all four benchmark clusters.

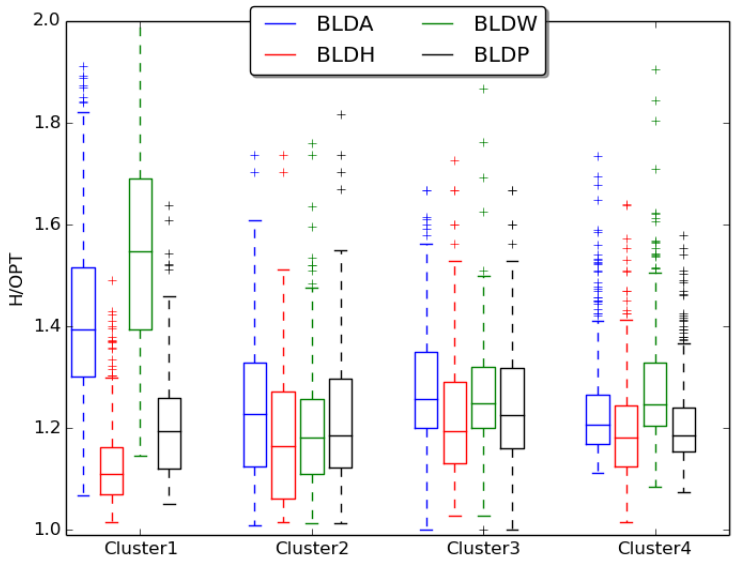


FIGURE 10.2: *The distribution of results returned during a preliminary study involving implementations of the BL algorithm of §3.2 with four different sorting strategies described above, when applied to the clustered benchmark data. The ratio H/OPT represents the packing accuracy, which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT. The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: Bottom-left algorithm (Cluster 1)				
Algorithm	BLDA	BLDH	BLDP	BLDW
BLDH	$< 1 \times 10^{-15}$	—		
BLDP	3.7×10^{-14}	1.2×10^{-9}	—	
BLDW	0.13	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.425 (3)	1.138 (1)	1.216 (2)	1.546 (3)

TABLE 10.5: *Comparison of four sorting strategies when applied to the bottom-left algorithm of §3.2 in respect of the first cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: Bottom-left algorithm (Cluster 2)				
Algorithm	BLDA	BLDH	BLDP	BLDW
BLDH	$< 1 \times 10^{-15}$	—		
BLDP	9.3×10^{-7}	3.7×10^{-14}	—	
BLDW	4.5×10^{-14}	4.8×10^{-8}	8.9×10^{-6}	—
Mean (Rank)	1.228 (4)	1.177 (1)	1.212 (3)	1.193 (2)

TABLE 10.6: *Comparison of four sorting strategies when applied to the bottom-left algorithm of §3.2 in respect of the second cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: Bottom-left algorithm (Cluster 3)				
Algorithm	BLDA	BLDH	BLDP	BLDW
BLDH	2.8×10^{-14}	—		
BLDP	2.0×10^{-5}	3.2×10^{-6}	—	
BLDW	0.087	1.6×10^{-12}	0.098	—
Mean (Rank)	1.313 (3)	1.259 (1)	1.288 (2)	1.317 (3)

TABLE 10.7: *Comparison of four sorting strategies when applied to the bottom-left algorithm of §3.2 in respect of the third cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: Bottom-left algorithm (Cluster 4)				
Algorithm	BLDA	BLDH	BLDP	BLDW
BLDH	4.4×10^{-14}	—		
BLDP	4.1×10^{-10}	0.002	—	
BLDW	0.057	$< 1 \times 10^{-15}$	2.6×10^{-14}	—
Mean (Rank)	1.239 (3)	1.196 (1)	1.213 (2)	1.28 (3)

TABLE 10.8: *Comparison of four sorting strategies when applied to the bottom-left algorithm of §3.2 in respect of the fourth cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

10.1.3 Results obtained by the IHR Algorithm

Boxplots of the results returned by the four IHR algorithmic implementations, when applied to the clustered benchmark data of Chapter 7, are shown in Figure 10.3. Applying the Friedman test to the results obtained for the various benchmark clusters yields p-values less than 1×10^{-15} for Clusters 1 and 4, a p-value 1.276×10^{-6} for Cluster 3, and a p-value 0.0017 for Cluster 2, indicating that there are statistically significant differences between the solutions achieved by some of the algorithmic implementations at a 5% level of significance. The presence of significant differences between the results returned by the four algorithmic implementations is clarified in the Nemenyi rank test results in Tables 10.9–10.12.

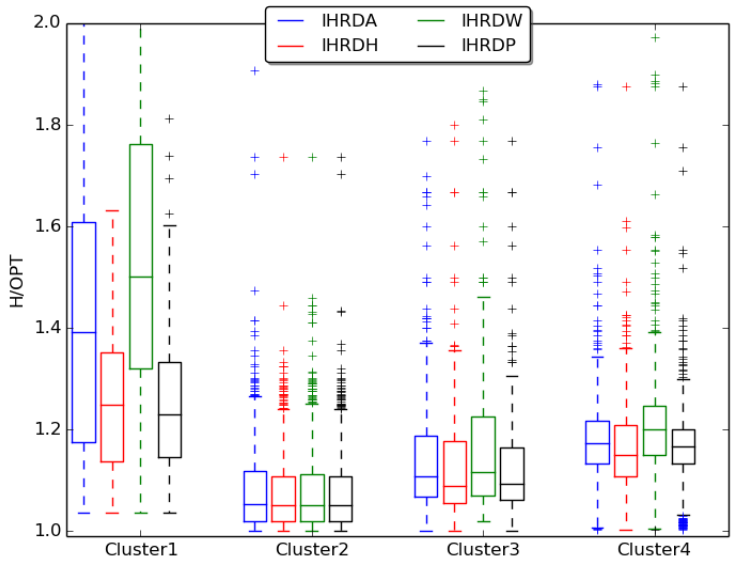


FIGURE 10.3: *The distribution of results returned during a preliminary study involving implementations of the IHR algorithm of §3.3 with four different sorting strategies described above, when applied to the clustered benchmark data. The ratio H/OPT represents the packing accuracy, which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT. The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

It is clear from these tables that the implementations employing the non-increasing height and perimeter sorting strategies are statistically similar in performance, returning favourable results in respect of all four benchmark clusters. The implementations employing the non-increasing area and width sorting strategies, on the other hand, performed the least effectively, ranked second and third, respectively, in respect of first and fourth benchmark clusters, while they are both ranked second in respect of second and third benchmark clusters.

Upon close inspection of the working of the IHR algorithm, described in §3.3, it is evident that the causes of the performance differences of the four algorithmic implementations lie in the recursion procedure of the algorithm as well as in the level restriction imposed during the packing process. By implementing the IHR algorithm in conjunction with the non-increasing height or perimeter sorting strategy, there is a significant chance that tall items are packed at the beginning of the procedure and that potential gaps within a level are later filled by short unpacked items if they fit in there. This results in a denser packing layout with relatively small gaps. By employing the other two sorting strategies, on the contrary, there is a risk that taller items contribute negatively to the packing height of a solution as they are only packed toward the end of the process.

p-values of the Nemenyi rank test: IHR algorithm (Cluster 1)				
Algorithm	IHRDA	IHRDH	IHRDP	IHRDW
IHRDH	2.3×10^{-10}	—		
IHRDP	1.4×10^{-11}	0.98	—	
IHRDW	2.8×10^{-5}	9.8×10^{-15}	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.4266 (2)	1.2605 (1)	1.2594 (1)	1.553 (3)

TABLE 10.9: *Comparison of four sorting strategies when applied to the improved heuristic recursive algorithm of §3.3 in respect of the first cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: IHR algorithm (Cluster 2)				
Algorithm	IHRDA	IHRDH	IHRDP	IHRDW
IHRDH	0.879	—		
IHRDP	0.253	0.69	—	
IHRDW	0.94	0.027	2.2×10^{-6}	—
Mean (Rank)	1.074 (2)	1.0712 (1)	1.0718 (1)	1.0775 (2)

TABLE 10.10: *Comparison of four sorting strategies when applied to the improved heuristic recursive algorithm of §3.3 in respect of the second cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: IHR algorithm (Cluster 3)				
Algorithm	IHRDA	IHRDH	IHRDP	IHRDW
IHRDH	0.01	—		
IHRDP	0.006	0.99	—	
IHRDW	0.805	0.0003	0.0002	—
Mean (Rank)	1.1967 (2)	1.1737 (1)	1.1702 (1)	1.2206 (2)

TABLE 10.11: *Comparison of four sorting strategies when applied to the improved heuristic recursive algorithm of §3.3 in respect of the third cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: IHR algorithm (Cluster 4)				
Algorithm	IHRDA	IHRDH	IHRDP	IHRDW
IHRDH	0.0002	—		
IHRDP	0.361	0.068	—	
IHRDW	1.3×10^{-8}	94.5×10^{-14}	1.9×10^{-13}	—
Mean (Rank)	1.1865 (2)	1.1681 (1)	1.1761 (1)	1.2151 (3)

TABLE 10.12: *Comparison of four sorting strategies when applied to the improved heuristic recursive algorithm of §3.3 in respect of the fourth cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

According to these results, the use of non-increasing height or non-increasing perimeter sorting strategy is statistically suggested in conjunction with the IHR algorithm with respect to all four benchmark clusters.

10.1.4 Results obtained by the BF Algorithm

As described in 3.4, the BF algorithm applies a dynamic rule in terms of which items are packed into a given strip. The algorithm first identifies the lowest available location in the strip, and then dynamically searches the list for the best-suited item to place in the location identified. Ties are resolved by packing the first item encountered in the list according to a pre-defined order. The four different sorting strategies described in §10.1 were each applied to the BF algorithm to sort the list of items at the beginning of the procedure. Boxplots of the results returned by these four algorithmic implementations, when applied to the clustered benchmark instances of Chapter 7, are shown in Figure 10.4.

Applying the Friedman test to the results obtained for the various clusters yields p-values less than 1×10^{-15} for Clusters 1, 2 and 4, and a p-value 9.82×10^{-12} for Cluster 3, indicating that there are statistically significant differences between the mean packing heights achieved by some of the algorithmic implementations at a 5% level of significance in all cases. The Nemenyi *post hoc* test was thus performed in order to determine between which pairs of implementation outputs statistical differences are discernible. Interestingly, it was found that the sorting strategy significantly influences the performance of the BF algorithm with respect to each cluster.

The implementation employing the decreasing perimeter sorting strategy achieved the smallest mean packing height solution with respect to the first cluster of benchmark instances. The implementation utilising the non-increasing height sorting strategy performed the second most effectively for this cluster, followed by the non-increasing area algorithmic implementation from which it did not differ significantly. In conjunction with the decreasing width sorting strategy, the BF algorithm packed the majority of instances in the first cluster to a relatively large packing height. These performance differences are statistically significant at a 5% level of significance, as shown in Table 10.13.

The above order of relative performances changed significantly in respect of the second cluster of data, as shown in Table 10.14. The performance of the implementation utilising the decreasing width sorting strategy improved dramatically in this case. While performing the worst in respect

of the first benchmark cluster, this algorithmic implementation returned the best results for the second cluster of benchmark instances. The implementation employing the decreasing perimeter sorting strategy became the second best performing implementation in this case and performed significantly similar to the non-increasing area algorithmic implementation. The reason for this is the characteristic of the benchmark instances belonging to the second cluster, which predominantly contains a large number of wide rectangular items. So selecting an item with the maximum width for packing in the available location during the packing process is more suitable and returns near-optimal results for these instances.

The comparison of the four algorithmic implementations in respect of the third cluster of data again yielded a different result. The implementation employing the decreasing area sorting strategy returned the most favourable result, achieving a mean packing height that is very close to the optimum height, outperforming all algorithms but the non-increasing perimeter algorithmic implementation, from which it did not differ significantly (see Table 10.15). This is explained by the fact that instances in this cluster are dominated by equally sized, narrow square items. Giving priority to larger items for packing would therefore be more beneficial in terms of minimising both wasted space and the eventual packing height.

The results obtained in respect of the fourth cluster of data were relatively similar to those of the third cluster. The implementation utilising the decreasing perimeter sorting strategy achieved the smallest mean packing height, followed by the non-increasing area implementation from which it did not differ significantly. The implementation employing the decreasing width sorting strategy performed poorly in respect of this cluster (see Table 10.16). Again the cause of the performance differences of the four algorithmic implementations in this case lies in the characteristic of the benchmark instances in this cluster. Since these instances are populated by equally sized, wide square items, a denser layout with a small packing height is achievable by the non-increasing perimeter implementation.

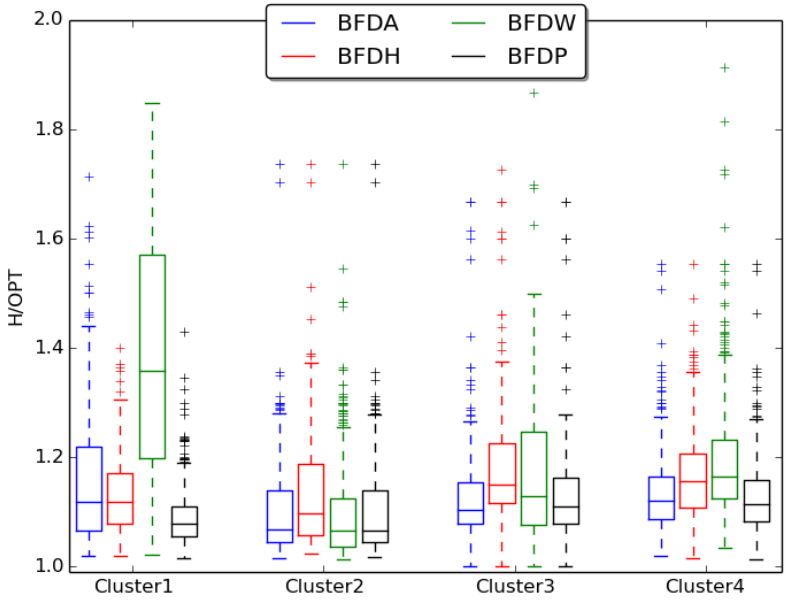


FIGURE 10.4: *The distribution of results returned during a preliminary study involving implementations of the BF algorithm of §3.4 with four different sorting strategies described above, when applied to the clustered benchmark data. The ratio H/OPT represents the packing accuracy, which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT. The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: Best-fit algorithm (Cluster 1)				
Algorithm	BFDA	BFDH	BFDP	BFDW
BFDH	0.93	—		
BFDP	2.1×10^{-9}	8.3×10^{-8}	—	
BFDW	2.9×10^{-14}	3.3×10^{-14}	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.167 (2)	1.138 (2)	1.1 (1)	1.382 (3)

TABLE 10.13: *Comparison of four sorting strategies when applied to the best-fit algorithm of §3.4 in respect of the first cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: Best-fit algorithm (Cluster 2)				
Algorithm	BFDA	BFDH	BFDP	BFDW
BFDH	3.8×10^{-14}	—		
BFDP	0.59	4.8×10^{-14}	—	
BFDW	4.4×10^{-5}	$< 1 \times 10^{-15}$	5.7×10^{-8}	—
Mean (Rank)	1.098 (2)	1.128 (3)	1.0981 (2)	1.095 (1)

TABLE 10.14: *Comparison of four sorting strategies when applied to the best-fit algorithm of §3.4 in respect of the second cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: Best-fit algorithm (Cluster 3)				
Algorithm	BFDA	BFDH	BFDP	BFDW
BFDH	5.1×10^{-9}	—		
BFDP	0.95	1.2×10^{-7}	—	
BFDW	0.00038	0.139	0.0028	—
Mean (Rank)	1.172 (1)	1.224 (2)	1.175 (1)	1.22 (2)

TABLE 10.15: *Comparison of four sorting strategies when applied to the best-fit algorithm of §3.4 in respect of the third cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: Best-fit algorithm (Cluster 4)				
Algorithm	BFDA	BFDH	BFDP	BFDW
BFDH	1.8×10^{-12}	—		
BFDP	0.23	4.6×10^{-14}	—	
BFDW	$< 1 \times 10^{-15}$	4.2×10^{-6}	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.138 (1)	1.169 (2)	1.131 (1)	1.194 (3)

TABLE 10.16: *Comparison of four sorting strategies when applied to the best-fit algorithm of §3.4 in respect of the fourth cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

According to this statistical analysis, the BF algorithm employing the non-increasing perimeter sorting strategy is statistically suggested for implementation in respect of the Clusters 1 and 4 benchmark instances. The BF algorithm employing the non-increasing area sorting strategy is, however, preferred for the benchmark instances in Cluster 2, while the non-increasing width sorting strategy should be selected as a sorting strategy in conjunction with the BF algorithmic implementation for the Cluster 3 data.

10.1.5 Results obtained by the CH Algorithm

Boxplots of the results returned by the four CH algorithmic implementations, when applied to the clustered benchmark data of Chapter 7, are shown in Figure 10.5. Applying the Friedman test to the results obtained for the various clusters yields p-values less than 1×10^{-15} for Clusters 1, 3, and 4, and a p-value 1.596×10^{-6} for Cluster 2, indicating that there are statistically significant differences between the solutions achieved by some of the algorithmic implementations at a 5% level of significance in all cases. The locations of significant differences between the results returned by the four algorithmic implementations are elucidated in the Nemenyi rank test results in Tables 10.17–10.20.

A significant difference in the performance of the four algorithmic implementations is observed at a 5% level of significance in respect of the first cluster of data (see Table 10.17). The non-increasing perimeter sorting strategy implementation outperformed the other three, achieving a packing height that is very close to the optimum height (an optimality gap of 8%) for this cluster. The non-increasing height sorting strategy implementation performed the second most effectively for this cluster, while the non-increasing width sorting strategy implementation achieved the worst mean packing height solutions.

The comparison of the four algorithmic implementations in respect of the second cluster yielded a different result. The performance of the implementation utilising the decreasing area sorting strategy improved in this case. While performing relatively poorly in respect of the first benchmark cluster, this algorithmic implementation returned favourable results for the second cluster of benchmark instances, outperforming all algorithmic implementations but the non-increasing perimeter algorithmic implementation, from which it did not differ statistically at a 5% level of significance (see Table 10.18). The implementation utilising the non-increasing height was ranked the last in respect of this cluster.

The results obtained in respect of the third cluster of data were different from those of the previous clusters. In this case, the three algorithmic implementations employing the non-increasing area, the non-increasing height, and the non-increasing perimeter sorting strategies did not differ statistically at a 5% level of significance, all achieving relatively small mean packing height solutions, as shown in Table 10.19. The implementation utilising the decreasing width sorting strategy performed poorly in respect of this cluster.

Comparison of the four algorithmic implementations in respect of the fourth benchmark cluster yielded relatively similar results as for the second cluster. The implementations employing the decreasing perimeter and the decreasing area sorting strategies did not differ statistically at a 5% level of significance, both achieving small mean packing height solutions in respect of this cluster, while the implementation utilising the decreasing width sorting strategy returned the worst results, as shown in Table 10.20.

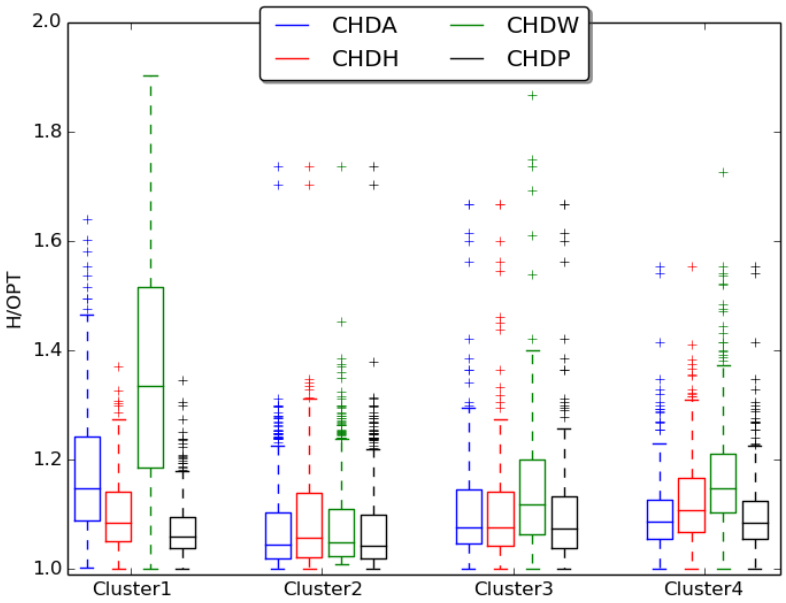


FIGURE 10.5: *The distribution of results returned during a preliminary study involving implementations of the CH algorithm of §3.5 with four different sorting strategies described above, when applied to the clustered benchmark data. The ratio H/OPT represents the packing accuracy, which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT. The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

Referring to the working of the CH algorithm, described in §3.5, it is evident that the use of an evaluation rule to select an item for packing, whereby ties are resolved using the ordered list, is the principal cause of the superior performance of this algorithm when implemented in conjunction with the non-increasing perimeter sorting strategy in respect of all clusters. The sorting strategy plays an important role during the packing process when two or more items achieve identical scores. In such a case, the CH algorithm packs the first item encountered in the list. Obviously, a denser layout with almost no waste would be achieved by the algorithm if it were to be implemented in conjunction with the non-increasing perimeter sorting strategy. Consequently, a packing height solution close to the optimum is returned by the algorithm in such a case. According to these results, the use of non-increasing perimeter sorting strategy is statistically suggested in conjunction with the CH algorithmic implementation with respect to all four benchmark clusters.

p-values of the Nemenyi rank test: Constructive heuristic (Cluster 1)				
Algorithm	CHDA	CHDH	CHDP	CHDW
CHDH	1.6×10^{-10}	—		
CHDP	4.0×10^{-14}	0.0006	—	
CHDW	3.3×10^{-12}	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.19 (3)	1.105 (2)	1.082 (1)	1.357 (4)

TABLE 10.17: *Comparison of four sorting strategies when applied to the constructive heuristic algorithm of §3.5 in respect of the first cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: Constructive heuristic (Cluster 2)				
Algorithm	CHDA	CHDH	CHDP	CHDW
CHDA	—			
CHDH	1.1×10^{-8}	—		
CHDP	0.97	6.6×10^{-10}	—	
CHDW	0.0035	0.046	0.0006	—
Mean (Rank)	1.073 (1)	1.09 (3)	1.071 (1)	1.081 (2)

TABLE 10.18: *Comparison of four sorting strategies when applied to the constructive heuristic algorithm of §3.5 in respect of the second cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: Constructive heuristic (Cluster 3)				
Algorithm	CHDA	CHDH	CHDP	CHDW
CHDH	0.963	—		
CHDP	0.199	0.067	—	
CHDW	8.7×10^{-9}	1.6×10^{-7}	5.9×10^{-14}	—
Mean (Rank)	1.15 (1)	1.149 (1)	1.142 (1)	1.202 (2)

TABLE 10.19: *Comparison of four sorting strategies when applied to the constructive heuristic algorithm of §3.5 in respect of the third cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

p-values of the Nemenyi rank test: Constructive heuristic (Cluster 4)				
Algorithm	CHDA	CHDH	CHDP	CHDW
CHDH	1.1×10^{-7}	—		
CHDP	0.87	1.0×10^{-9}	—	
CHDW	$< 1 \times 10^{-15}$	4.1×10^{-14}	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.104 (1)	1.129 (2)	1.102 (1)	1.172 (3)

TABLE 10.20: *Comparison of four sorting strategies when applied to the constructive heuristic algorithm of §3.5 in respect of the fourth cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances). The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

10.2 Comparison of Strip Packing Heuristics

The relative performances of the five heuristics described in Chapter 3 are compared in this section in terms of both solution quality and execution time with one another in respect of the four benchmark clusters of Chapter 7. The comparison takes place in respect of the best implementation of each heuristic, as identified during the computational study of §10.1. Interestingly, it was found that the order of the relative performances of the five algorithms varies significantly, depending on the benchmark cluster under investigation. This is clear from boxplots of these relative performances shown in Figure 10.6, and is elucidated by the Nemenyi rank test results in Tables 10.21–10.24.

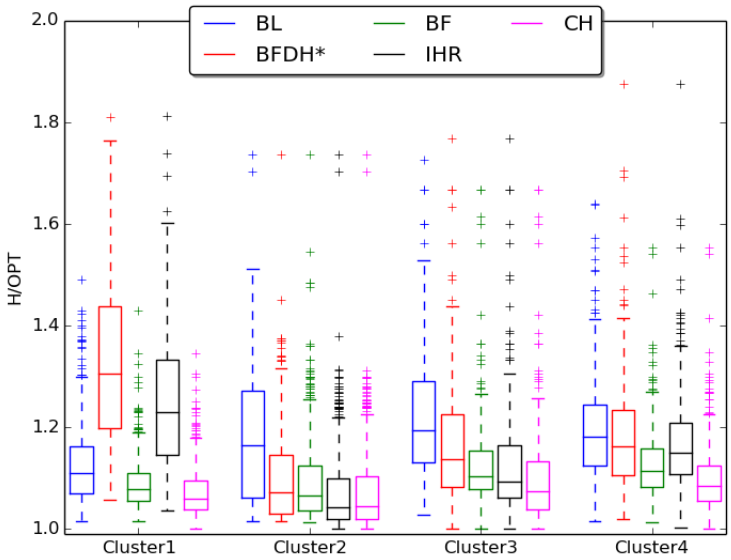


FIGURE 10.6: *The distribution of best results returned by the five heuristics described in Chapter 3, when applied to the clustered benchmark data of Chapter 7. The ratio H/OPT represents the packing accuracy in (9.1), which is the ratio between the mean strip height H achieved by an algorithm and the optimal height (or the appropriate lower bound) OPT.*

The performance differences between the five algorithms are noticeable and significant with respect to the first cluster of data (see Table 10.21). The CH algorithm performed the best in

p-values of the Nemenyi rank test: SPP heuristic algorithms (Cluster 1)					
Algorithm	BFDH*	BL	IHR	BF	CH
BL	7.8×10^{-15}	—			
IHR	0.0002	1.4×10^{-11}	—		
BF	$< 1 \times 10^{-15}$	6.1×10^{-5}	$< 1 \times 10^{-15}$	—	
CH	$< 1 \times 10^{-15}$	1.2×10^{-12}	$< 1 \times 10^{-15}$	0.03	
Mean (Rank)	1.336 (5)	1.138 (3)	1.259 (4)	1.1 (2)	1.08 (1)

TABLE 10.21: *Comparison of the results returned by the five SPP heuristic algorithms, described in Chapter 3, in respect of the first cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the algorithms, with their ranks shown in parentheses (a rank of 1 indicates that the algorithm achieved the smallest mean packing height for the instances in the cluster of benchmark data).*

p-values of the Nemenyi rank test: SPP heuristic algorithms (Cluster 2)					
Algorithm	BFDH*	BL	IHR	BF	CH
BL	$< 1 \times 10^{-15}$	—			
IHR	6.7×10^{-14}	$< 1 \times 10^{-15}$	—		
BF	0.0003	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—	
CH	5.1×10^{-14}	$< 1 \times 10^{-15}$	0.998	$< 1 \times 10^{-15}$	
Mean (Rank)	1.099 (3)	1.1774 (4)	1.0712 (1)	1.095 (2)	1.0716 (1)

TABLE 10.22: *Comparison of the results returned by the five SPP heuristic algorithms, described in Chapter 3, in respect of the second cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the algorithms, with their ranks shown in parentheses (a rank of 1 indicates that the algorithm achieved the smallest mean packing height for the instances in the cluster of benchmark data).*

p-values of the Nemenyi rank test: SPP heuristic algorithms (Cluster 3)					
Algorithm	BFDH*	BL	IHR	BF	CH
BL	1.2×10^{-6}	—			
IHR	5.8×10^{-5}	5×10^{-14}	—		
BF	0.038	6.6×10^{-14}	0.429	—	
CH	4.5×10^{-14}	$< 1 \times 10^{-15}$	0.0004	7.8×10^{-8}	
Mean (Rank)	1.214 (3)	1.259 (4)	1.17 (2)	1.172 (2)	1.142 (1)

TABLE 10.23: *Comparison of the results returned by the five SPP heuristic algorithms, described in Chapter 3, in respect of the third cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the algorithms, with their ranks shown in parentheses (a rank of 1 indicates that the algorithm achieved the smallest mean packing height for the instances in the cluster of benchmark data).*

p-values of the Nemenyi rank test: SPP heuristic algorithms (Cluster 4)					
Algorithm	BFDH*	BL	IHR	BF	CH
BL	0.02	—			
IHR	1.00	0.03	—		
BF	1.0×10^{-13}	5.1×10^{-14}	9.1×10^{-14}	—	
CH	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	2.6×10^{-10}	
Mean (Rank)	1.1844 (3)	1.1969 (4)	1.1681 (3)	1.1317 (2)	1.1028 (1)

TABLE 10.24: *Comparison of the results returned by the five SPP heuristic algorithms, described in Chapter 3, in respect of the fourth cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the algorithms, with their ranks shown in parentheses (a rank of 1 indicates that the algorithm achieved the smallest mean packing height for the instances in the cluster of benchmark data).*

respect of this cluster, significantly outperforming the other four algorithms at a 5% level of significance. The BF algorithm yielded fairly good results, ranked second for this cluster, and outperformed the BL and IHR algorithms. In contrast to the favourable results achieved by the CH algorithm, the BFDH* algorithm performed poorly in respect of this cluster. The mean packing height achieved by the latter algorithm is 25% worse than that achieved by the CH algorithm.

In respect of the second cluster of data, the order of the relative performances of the five algorithms changed significantly. The performance of the IHR algorithm improved dramatically in this case. Whereas it was the second-worst performing algorithm in respect of the benchmark instances in Cluster 1, the IHR algorithm became the best performing algorithm in respect of the Cluster 2 data, outperforming all algorithms but the CH algorithm, from which it did not differ statistically at a 5% level of significance (see Table 10.22). The next best algorithm in this case was the BF algorithm. The BL algorithm was the worst performing algorithm for this cluster, and was significantly outperformed by the BFDH* algorithm.

A significant difference in the performance of the five algorithms was also observed at a 5% level of significance in respect of the third cluster of data (see Table 10.23). The CH algorithm returned the best results for this cluster, achieving the smallest mean packing height. The IHR and BF algorithms were the next-best performing algorithms in this case, but did not return results differing significantly from one another. The BFDH* and BL algorithms were ranked respectively the third and worst performing algorithms in this case.

As was the case for the third cluster, the CH algorithm was the best performing algorithm in respect of the fourth benchmark cluster, packing the majority of these benchmark instances to a relatively low packing height. The BF algorithm performed the second most effectively in this case, significantly outperforming the IHR algorithm. The BL algorithm was again ranked the worst performing algorithm for this cluster (see Table 10.24).

The natural logarithms of the execution times of the five algorithms are shown in Figure 10.7. The BFDH* algorithm is more efficient in terms of computation time, while the time required to execute the IHR algorithm is relatively high. Nonetheless, all five algorithms are relatively fast — the average time required to solve large problem instances was between 1 and 20 seconds.

According to these results, the CH algorithm performs well on average. The use of the scoring rule to select the appropriate item to pack during the packing process is the principal advantage of this algorithm. This property enables the algorithm to identify the best item to fill any

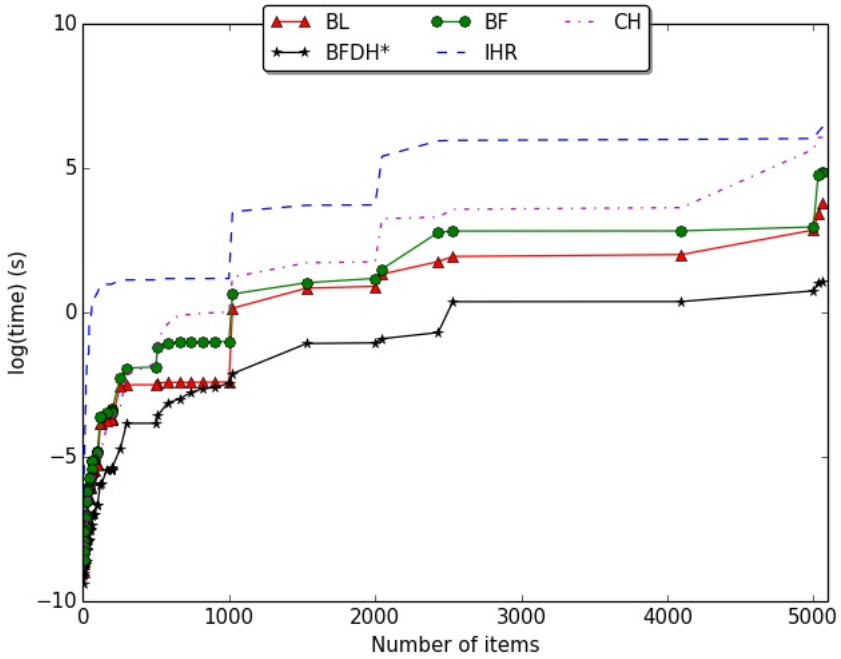


FIGURE 10.7: *Execution time of the five heuristics described in Chapter 3 as a function of the number of items per SPP instance.*

available space during the packing process, thereby allowing it to achieve a denser layout. This packing rule is particularly important when packing the set of instances belonging to the first cluster. Since these instances are populated by long and flat, or tall and thin, rectangular items, the sliding method employed in the BL algorithm or the level restriction pertaining to the IHR and BFDH* algorithms may leave considerable gaps in the packing layout as unused regions within a level may not be filled and taller rectangles may therefore contribute significantly to the overall packing height of a solution. Moreover, the dynamic search strategy involved in the BF algorithm enables the algorithm to generate a dense layout, although this is not enough to guarantee good results.

A comparison between the five heuristics in respect of the second benchmark cluster, however, showed that the IHR algorithm performed the best, competing with the CH algorithm in this case. The reason for this is the characteristic of the benchmark instances belonging to the second cluster, which predominantly contains a large number of wide rectangular items. Such a set of items can be effectively packed in levels according to the recursive method employed by the IHR algorithm. A packing solution which contains a relatively small area of empty spaces within each level, and therefore with a relatively small packing height, is generated accordingly.

In respect of the benchmark instances in Clusters 3 and 4, the CH algorithm is still preferable and is statistically suggested as an effective packing algorithm for these instances.

In [164, 165], the IHR algorithm was reported to be superior to the BF algorithm. According to the above results, however, this claim only holds for benchmark instances in Clusters 2 and 3 (see Figure 10.6 and Tables 10.21–10.24). Moreover, the BL algorithm has been shown to be inferior to the BF and IHR algorithms [29, 165]. The fact that the BF is superior to the BL algorithm is confirmed here, but the IHR algorithm only outperforms the BL algorithm for benchmark instances in Clusters 2, 3 and 4. Finally, the computational results in [115] show that the CH algorithm returns favourable results for large-scale problem instances and performs well on average. The results obtained above corroborate the latter claim, but the CH algorithm

was shown to be inferior to the IHR algorithm for some large benchmark instances in Cluster 2, thus contradicting the former claim.

Accordingly, it is advisable to take into consideration the underlying characteristics of the benchmark instances employed during a comparative algorithmic study in order to avoid biased conclusions with respect to the algorithms' relative effectiveness.

10.3 Chapter Summary

In this chapter, the results returned by the five heuristics reviewed in Chapter 3, when applied to the clustered benchmark data of Chapter 7, were presented and interpreted. A presentation of the results returned by each of the heuristics in the form of boxplots together with the non-parametric Friedman and the Nemenyi *post hoc* tests applied to these results were provided in §10.1. The same tests were performed in §10.2 to compare the relative performances of the five heuristics in respect of the clustered benchmark instances of Chapter 7.

A summary of the most effective algorithmic implementations for the five heuristics with respect to each cluster of benchmark instances may be found in Table 10.25. This table indicates which sorting strategy should be implemented in conjunction with each algorithm, as well as the best algorithm for packing instances in each cluster.

Algorithms	Cluster 1	Cluster 2	Cluster 3	Cluster 4
BFDH*	DH	DH	DH	DH
BL	DH	DH	DH	DH
IHR	DP	DH	DP	DH
BF	DP	DW	DA	DP
CH	DP	DP	DP	DP
Best algorithm	CHDP	IHRDH	CHDP	CHDP

TABLE 10.25: *Summary of the best performing heuristic implementations with respect to each cluster of benchmark instances. The extensions 'DA', 'DH', 'DP', and 'DW' stand for 'decreasing area', 'decreasing height', 'decreasing perimeter', and 'decreasing width', respectively.*

The results of the comparative study of this chapter first show that the packing order has a crucial effect on the performance of the different heuristics. This is in line with what has been reported in the literature [29, 91, 115, 116]. The results obtained in this study also demonstrate that the characteristics of the benchmark instances affect the mean solution quality achieved by the various packing algorithms. This has not, however, been taken into account in the literature prior to this study. Comparisons of different algorithmic approaches often take place in respect of a specific set of data, which do not sufficiently represent the wide range of benchmark problem instances available in SPP repositories. Moreover, the effects of the characteristics of the benchmark instances on the performances of various algorithms have not been studied. It is, therefore, recommended to take into consideration the underlying characteristics of the benchmark instances employed during a comparative algorithmic study in order to avoid biased conclusions with respect to the relative effectiveness of SPP algorithms.

CHAPTER 11

Efficient Implementation of Known Hybrid Metaheursitics

Contents

It is evident from the literature review of Chapter 4 that the majority of metaheuristic SPP solution techniques are hybrid algorithms in which a metaheuristic approach (such as a GA or the method of SA) is combined with a heuristic placement routine. In these hybrid approaches, the task of the metaheuristic is to search for an order in which the items should be packed, while the second algorithm is required to evaluate the quality of the packing permutation and also to transform it into a packing layout. Results from the literature indicate that the performance of the hybrid algorithms depends strongly on the nature of the decoding procedure and the problem dimension. Moreover, the choice of certain parameter values in the metaheuristic technique employed also influences the performances of these hybrid algorithms significantly [91, 116].

The main objective of this chapter is to identify superior implementations of a GA and the method of SA employed in the hybrid GA and hybrid SA SPP algorithms, in terms of their constituent elements. More specifically, the aim is to study various parameter settings of each metaheuristic algorithm in order to select the most suitable hybrid GA and hybrid SA implementations for the purposes of the comparative study carried out in the following chapter. A limited computational study based on an experimental design and a sensitivity analysis was conducted accordingly. Details of these experimental studies, as well as the results thus obtained, are reported and interpreted in this chapter. The CH algorithm described in §3.5 was implemented as decoding routine for both hybrid techniques in this preliminary study, because it was found to be the best performing SPP heuristic in Chapter 10.

The chapter opens in §11.1 with a description of the method of analysis performed. Thereafter, details of the experimental design followed and the results obtained in search of the best implementation of a hybrid GA are presented in §11.2. This is followed by the presentation of a similar computational study in respect of a hybrid SA implementation in §11.3. A summary of the contents of the chapter is finally provided in §11.4.

11.1 Method of Analysis

A number of generic decisions have to be made during the implementation of the metaheuristic search algorithms considered in this dissertation. In the case of a hybrid GA, these generic decisions concern the method of generating the initial population, the size of the initial population, the number of generations considered, the types of operators applied to explore and exploit the search space, and the probabilities according to which the various algorithmic operators, such as crossover and mutation, are applied. The generic choices for a suitable implementation of the hybrid SA algorithm, on the other hand, are typically summarised in its cooling schedule. This includes the selection of an appropriate initial temperature value, the determination of a relevant cooling schedule, and the choice of a suitable epoch management policy. Different choices of the operators and parameter values mentioned above may affect the performance of the algorithms.

The preliminary study conducted in this chapter consists of testing various parameter combinations of each metaheuristic technique implementation, with the specific aim of identifying the most efficient implementation of each. In the case of the hybrid GA, different types of crossover operators, selection procedures and replacement techniques are combined and compared with one another. Different values of the crossover rate are also evaluated. In the case of the hybrid SA algorithm, various values of the initial temperature, the parameter employed in the cooling schedule, and the length of an epoch during the search are tested and compared. All tests and comparisons are carried out in respect of the clustered benchmark instances of Chapter 7, and the results are represented in the form of boxplots together with the appropriate statistical analyses, as described in §9.1.

11.2 Selection of the best Hybrid GA Implementation

In order to obtain the best results achievable by the method of hybrid GA, a suitable combination of algorithmic parameter values has to be selected. The GA parameter optimisation experiments considered in this computational study consist of two phases, whereby two sets of parameters are considered separately. The set of parameters considered during the first phase are the crossover operators, the selection procedures, and the replacement techniques. During the second phase of the experiment, the best combinations of parameter values obtained during the first phase are adopted as constant values. The set of parameters varied during the second phase of the experiment are the crossover rate parameter and the fitness function. Some parameters remain constant during the experimental study. These are the mutation probability parameter and the generation size parameter.

11.2.1 Experimental Design

The first analysis carried out in the aforementioned experimental design consists of eight experimentations. More specifically, the hybrid GA combined with the CH algorithm is implemented

in eight different incarnations based on two different types of crossover operators, two types of selection procedures, and two types of replacement techniques. The best output obtained during the first analysis is further subjected to a sensitivity analysis during the second phase analysis. During this phase, three values of the crossover probability parameter and two different types of fitness function are evaluated. The details of these experimental studies are described in this section.

First experimental study

The eight different algorithmic incarnations considered during the first experimental study are summarised in Table 11.1. PMX crossover [70] combined with SUS selection [12] and elitism replacement [50] is considered during the first experiment, denoted by *HGA1*. The second experiment, denoted by *HGA2*, involves a combination of the PMX crossover operator with tournament selection and elitism replacement, while the third experiment, denoted by *HGA3*, is a combination of CX crossover with SUS selection and elitism replacement. CX crossover [130], together with elitism replacement and tournament selection, constitutes the fourth experiment, denoted by *HGA4*, while PMX crossover combined with SUS selection and steady state replacement [68] constitutes the fifth experiment, denoted by *HGA5*. The sixth experiment, denoted by *HGA6*, involves a combination of PMX crossover, tournament selection and steady state replacement, while the seventh experiment, denoted by *HGA7*, is concerned with a combination of CX crossover, SUS selection and steady state replacement. In the last experiment, denoted by *HGA8*, CX crossover is combined with tournament selection and steady state replacement.

	Elitism replacement		Steady state replacement	
	PMX crossover	CX crossover	PMX crossover	CX crossover
SUS selection	*HGA1*	*HGA3*	*HGA5*	*HGA7*
Tournament selection	*HGA2*	*HGA4*	*HGA6*	*HGA 8*

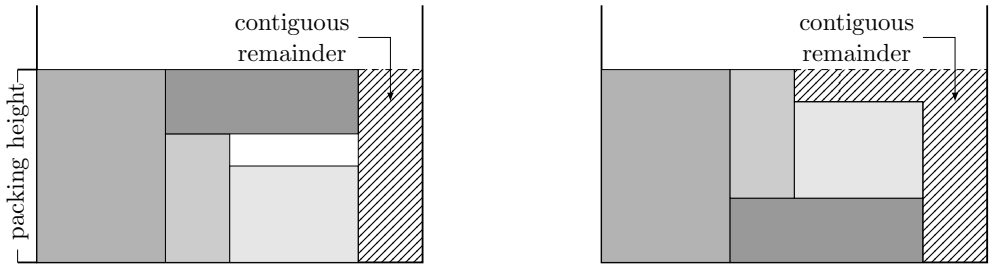
TABLE 11.1: *Eight different incarnations of the hybrid GA combined with the CH algorithm, based on different types of the algorithmic operators, during the first experimental study. The table entries are the names assigned to the experiments.*

During this first experimental study, the GAs are implemented using a population size of 50. The crossover probability is selected as 60% and the mutation probability as 3%. These values have previously been utilised in experimentations by Hopper and Turton [91], and also by Burke *et al.* [30]. Since the initial ordering of the input permutations has an effect on the performance of the algorithms [5, 116], as demonstrated for the underlying CH decoding algorithm in Chapter 8, the initial population is seeded with four permutations comprising items sorted according to decreasing height, width, area, and perimeter, respectively. The initial population is generated randomly and contains the seeded individuals. The swap mutation is employed to diversify the search and the packing solution is evaluated in terms of the packing height returned.

Second experimental study

The second experimental study consists of studying the effect of changing the inputs of the genetic algorithmic search, such as the crossover rate and the fitness function, on the performance of the best performing hybrid GA algorithm, as identified during the first experimental study,

in respect of the clustered benchmark instances of Chapter 7. Two studies are carried out. In the first case, the best performing outputs obtained during the first experimental study are implemented using a different fitness function: The packing solution is evaluated in terms of the area of contiguous remainder of the packing pattern instead of merely the packing height returned. This alternative fitness function may be used to resolve ties if two packing solutions yield the same height, while one of the packings achieves a more desirable layout (*e.g.* the second packing π_2 in Figure 11.1). The area of contiguous remainder of a packing pattern is defined as the unused area below the skyline delimited by the last items packed, as illustrated by the dashed areas in Figure 11.1.



(a) *Packing* π_1 (b) *Packing* π_2

FIGURE 11.1: *The contiguous remainders of two packing patterns. The regions delimited by dashed lines represent the areas of contiguous remainder.*

During the genetic algorithmic search in this case, packing solutions are compared in terms of the value of their underlying area of contiguous remainder as a measure of their fitness. The packing solution with a larger fitness value is preferred as it achieves a more dense packing layout. The packing height of the best packing solution is returned at the end of the search.

In the second case study, a sensitivity analysis is performed in respect of the crossover rate. Two other crossover probabilities are considered: The hybrid GA is also implemented for crossover probabilities of 90% and 80%. These implementations are executed five times, terminating a run when 5 000 iterations have been executed. A time limit of 60 seconds is additionally imposed for large instances that contain more than 5 000 items. The best solution returned is recorded in each case.

All algorithms were coded in Python using Spyder Version 2.7.6. The algorithms were all executed on an Intel Core i7-4790 CPU running at 3.60 GHz with 8 GB RAM in the Ubuntu 14.04 operating system.

11.2.2 Computational Results

The numerical results obtained when following the experimental design described in §11.2.1 are presented in this section. First, the results returned by the eight hybrid GA implementations during the first experimental study are compared in terms of their solution quality. Thereafter, the same comparison is carried out in respect of the algorithms implemented during the second experimental study. All the results are reported in the form of boxplots substantiated by appropriate statistical analyses. All significance tests were performed utilising appropriate **R** functions within the RStudio software environment, and a significance level of $\alpha = 0.05$ was adopted for all the results presented here.

First experimental study results

Boxplots of the results obtained by the eight implementations described in Table 11.1, when applied to the clustered benchmark instances of Chapter 7, are shown in Figure 11.2. For each cluster, a similar trend emerges with respect to the relative performances of the implementations, as can be observed from the boxplots. More precisely, the four implementations HGA5, HGA6, HGA7, and HGA8, which employ the steady state technique as a replacement operator, packed the majority of benchmark instances to a larger relative packing height than did the other four implementations, independently of the clusters. This observation is supported by the Friedman test applied to the results for all the benchmark clusters, yielding a p-value less than 1×10^{-15} in each case, and by the Nemenyi test for which the p-values are provided in Tables 11.2–11.5. The relatively poor performances of these four algorithmic incarnations may be explained by the fact that steady state replacement allows only one member of the generation to be replaced at a time during the genetic search procedure, which thus restricts the level of diversification achievable during the search.

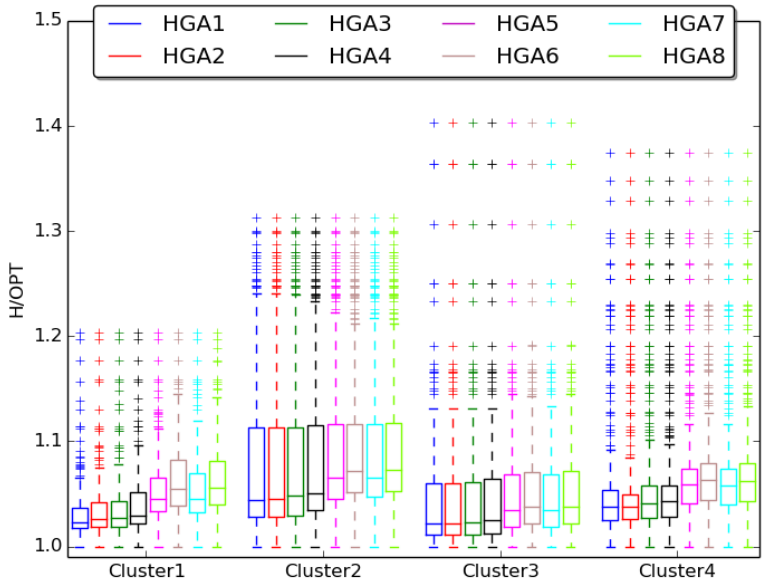


FIGURE 11.2: *The distribution of results returned during the eight experiments involving a combination of the Hybrid GA implementations of Table 11.1 with the CH algorithm as decoding procedure, as described in §11.2.1, when applied to the clustered benchmark data. The ratio H/OPT represents the packing accuracy, which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT.*

Performing the Nemenyi test in respect of the eight algorithmic solution sets for the first benchmark cluster suggests that there is a statistical difference at a 5% level of significance between the results returned by the group of algorithms that employ different replacement strategies, but that there is no statistical difference at a 5% level of significance between the results of algorithms that utilise the same replacement techniques, except between the HGA1 and HGA4 implementations (see Table 11.2). The group of implementations employing an elitism replacement technique achieved larger mean packing heights than did the other group of implementations employing a steady state replacement. Accordingly, the elitism strategy is the best choice as replacement method in the implementation of hybrid GAs for the first cluster. The hybrid GA algorithm is not sensitive with respect to crossover operators and selection procedures implemented for this cluster, and so a combination of PMX crossover and SUS selection is selected arbitrarily as configuration of the hybrid GA for the Cluster 1 benchmark instances in this study.

p-values of the Nemenyi rank test: Hybrid GA combined with the CH algorithm (Cluster 1)								
Algorithm	HGA1	HGA2	HGA3	HGA4	HGA5	HGA6	HGA7	HGA8
HGA2	0.933	—						
HGA3	0.652	0.99	—					
HGA4	0.0015	0.0903	0.315	—				
HGA5	7.3×10^{-14}	7.1×10^{-14}	5.8×10^{-14}	1.0×10^{-9}	—			
HGA6	$< 1 \times 10^{-15}$	7.6×10^{-14}	7.3×10^{-14}	6.6×10^{-13}	0.97	—		
HGA7	9.3×10^{-14}	1.5×10^{-13}	3.1×10^{-12}	6.5×10^{-6}	0.836	0.208	—	
HGA8	$< 1 \times 10^{-15}$	7.8×10^{-14}	7.4×10^{-14}	1.8×10^{-13}	0.897	1.00	0.108	—
Mean (Rank)	1.0407 (1)	1.0435 (1)	1.0444 (1)	1.048 (1)	1.0607 (2)	1.0694 (2)	1.0617 (2)	1.0707 (2)

TABLE 11.2: *Comparison of the results returned by the eight hybrid GA implementations in Table 11.1 combined with the CH algorithm as decoding procedure in respect of the first cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

p-values of the Nemenyi rank test: Hybrid GA combined with the CH algorithm (Cluster 2)								
Algorithm	HGA1	HGA2	HGA3	HGA4	HGA5	HGA6	HGA7	HGA8
HGA2	0.214	—						
HGA3	0.689	0.995	—					
HGA4	0.0069	0.94	0.53	—				
HGA5	1.0×10^{-13}	8.3×10^{-10}	3.7×10^{-12}	1.2×10^{-6}	—			
HGA6	$< 1 \times 10^{-15}$	6.5×10^{-14}	7.9×10^{-14}	6.2×10^{-14}	0.02	—		
HGA7	9.0×10^{-14}	1.4×10^{-7}	1.1×10^{-9}	8.1×10^{-5}	0.99	0.00095	—	
HGA8	$< 1 \times 10^{-15}$	7.8×10^{-14}	$< 1 \times 10^{-15}$	7.0×10^{-14}	0.0017	0.997	4.4×10^{-5}	—
Mean (Rank)	1.083 (1)	1.0829 (1)	1.0846 (1)	1.0867 (1)	1.0941 (2)	1.0983 (2)	1.0951 (2)	1.0989 (2)

TABLE 11.3: *Comparison of the results returned by the eight hybrid GA implementations in Table 11.1 combined with the CH algorithm as decoding procedure in respect of the second cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

p-values of the Nemenyi rank test: Hybrid GA combined with the CH algorithm (Cluster 3)								
Algorithm	HGA1	HGA2	HGA3	HGA4	HGA5	HGA6	HGA7	HGA8
HGA2	0.99	—						
HGA3	0.99	1.00	—					
HGA4	0.98	0.99	0.99	—				
HGA5	1.8×10^{-5}	0.0001	0.0001	0.0013	—			
HGA6	6.9×10^{-11}	1.7×10^{-9}	1.3×10^{-9}	3.2×10^{-8}	0.46	—		
HGA7	0.0018	0.01	0.009	0.0469	0.974	0.0495	—	
HGA8	1.1×10^{-10}	2.6×10^{-9}	2.0×10^{-9}	4.8×10^{-8}	0.503	1.00	0.059	—
Mean (Rank)	1.0984 (1)	1.0982 (1)	1.0989 (1)	1.0999 (1)	1.1057 (2)	1.1085 (2)	1.1055 (2)	1.1091 (2)

TABLE 11.4: *Comparison of the results returned by the eight hybrid GA implementations in Table 11.1 combined with the CH algorithm as decoding procedure in respect of the third cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

p-values of the Nemenyi rank test: Hybrid GA combined with the CH algorithm (Cluster 4)								
Algorithm	HGA1	HGA2	HGA3	HGA4	HGA5	HGA6	HGA7	HGA8
HGA2	1.00	—						
HGA3	0.91	0.73	—					
HGA4	0.81	0.59	1.00	—				
HGA5	7.3×10^{-14}	7.4×10^{-14}	1.0×10^{-13}	5.9×10^{-14}	—			
HGA6	7.0×10^{-14}	4.3×10^{-14}	7.2×10^{-14}	8.9×10^{-14}	1.00	—		
HGA7	6.4×10^{-14}	1.0×10^{-13}	2.1×10^{-12}	1.0×10^{-11}	0.76	0.52	—	
HGA8	7.3×10^{-14}	8.1×10^{-14}	9.2×10^{-14}	5.5×10^{-14}	1.00	1.00	0.68	—
Mean (Rank)	1.0587 (1)	1.0579 (1)	1.0625 (1)	1.0622 (1)	1.0759 (2)	1.0793 (2)	1.0747 (2)	1.0787 (2)

TABLE 11.5: *Comparison of the results returned by the eight hybrid GA implementations in Table 11.1 combined with the CH algorithm as decoding procedure in respect of the fourth cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

Performing the same significance tests in respect of the results returned for the second, the third and the fourth clusters of benchmark instances yield analogous results as for the previous cluster. The four implementations that employ an elitism replacement strategy outperformed the other four, achieving packing heights that are very close to the optimum height (with an optimality gap of approximately 8% for the Cluster 2, 9% for the Cluster 3 data, and between 5% and 6% for the Cluster 4 benchmark instances), as shown in Tables 11.3–11.5. This indicates that an implementation achieving the smallest packing height most consistently for Clusters 2, 3 and 4 is likely to be an implementation which utilises the elitism replacement strategy. Of the four implementations that conform to this restriction, the HGA2 implementation is arbitrarily selected as hybird GA algorithmic implementation in the comparative study carried out later in this dissertation.

Summarising the results in Tables 11.2–11.5, a hybrid GA implementation employing elitism replacement is statistically suggested for implementation in respect of all four benchmark clusters. Moreover, the hybrid GA is not sensitive with respect to the crossover operator and selection technique employed. It is acknowledged that the performance of a hybrid GA algorithm depends sensitively on the decoding method employed. The use of different decoding algorithms may indeed lead to different results. In this preliminary study, however, the CH algorithm was chosen as decoding method throughout for two reasons: It is one of the latest proposed strip packing heuristics and it performs very well, on average, as reported in Chapter 10.

One may also notice that the boxplots in Figure 11.2 contain outliers that are practically the same for all algorithms and for each cluster. These are the solutions returned by the algorithms when applied to the instances in the class of non-zero waste instances for which the respective optimal solutions are not known. The optimal solution may, in fact, only be achievable if rotation of items is allowed. Since rotation of items is not allowed in this study, the hybrid GA algorithms may generate, for a certain number of iterations, solutions that yield the same packing height but with different packing layouts.

Second experimental study results

Based on the first experimental study results, the HGA1 implementation (a hybrid GA implementation employing elitism replacement combined with PMX crossover and SUS selection) is the best performing hybrid GA implementation for the SPP instances in Cluster 1, while the

HGA2 implementation (a hybrid GA implementation employing elitism replacement together with PMX crossover and tournament selection) is the most suitable hybrid GA implementation for benchmark data in Clusters 2 to 4. These algorithmic implementations were thus considered during the second experimental study.

Boxplots of the results returned by the aforementioned hybrid GAs, when implemented using different fitness functions as described in §11.2.1, applied to the clustered benchmark instances are shown in Figure 11.3. It is clear from these boxplots that the orders of the comparative performances of the two algorithmic incarnations that were found to be superior during the first experimental study remain the same for each cluster. The hybrid GA implementation employing packing height as fitness function, however, yielded better packing solutions for all the clusters than does the other implementation utilising the area of contiguous remainder as fitness function. This observation is supported by applying the Friedman test to the results for the various clusters, each yielding a p-value less than 1×10^{-15}.

These results are not surprising, as they are direct implications of the use of the area of contiguous remainder as a fitness function in the algorithmic implementation. At some stage of the genetic search, the population of solutions may contain a large number of packing patterns that achieve dense layouts, but which are large packing height solutions. There is thus a significant chance that the implementation returns a packing solution achieving an unfavorable packing height at the end of the search. The use of packing height as fitness function is therefore recommended in the algorithmic implementation of the hybrid GA.

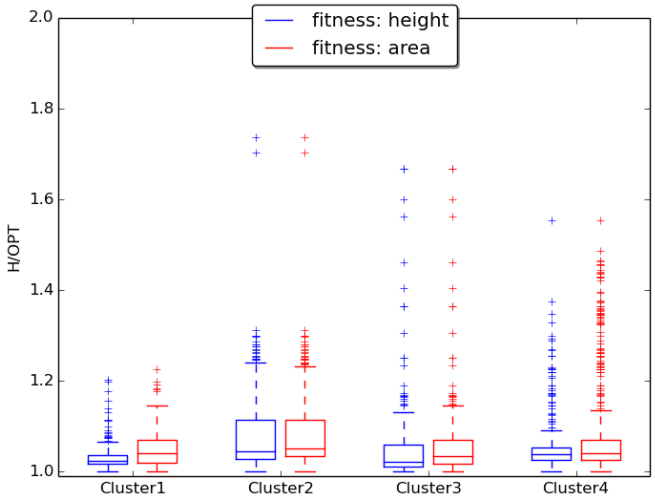


FIGURE 11.3: *The distribution of results returned during a sensitivity analysis in respect of the fitness function employed within the best performing Hybrid GA implementations of the first experimental study, namely a hybrid GA implementation employing elitism replacement combined with PMX crossover and SUS selection for the benchmark data in Cluster 1, and a hybrid GA implementation employing elitism replacement together with PMX crossover and tournament selection for the SPP instances in Clusters 2 to 4. The ratio H/OPT represents the packing accuracy, which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT. Results returned by the Hybrid GA implementations employing the packing height as fitness function are represented by blue boxplots, while the results are represented by red boxplots for the implementation employing area of contiguous remainder as fitness function.*

When studying the effect of varying the value of the crossover rate on the performance of the hybrid GA algorithm in respect of the clustered benchmark instances, no obvious differences are observed as there is not a single algorithmic implementation that performs the best overall for all four clusters. The boxplots of the corresponding results in Figure 11.4 do not show

a clear difference between the solutions obtained by the three algorithmic incarnations. This observation is supported by applying the Friedman test to the results obtained for the various clusters, yielding p-values between 0.16 and 0.91, suggesting that the differences in average packing height are not statistically significant at a 5% level of significance.

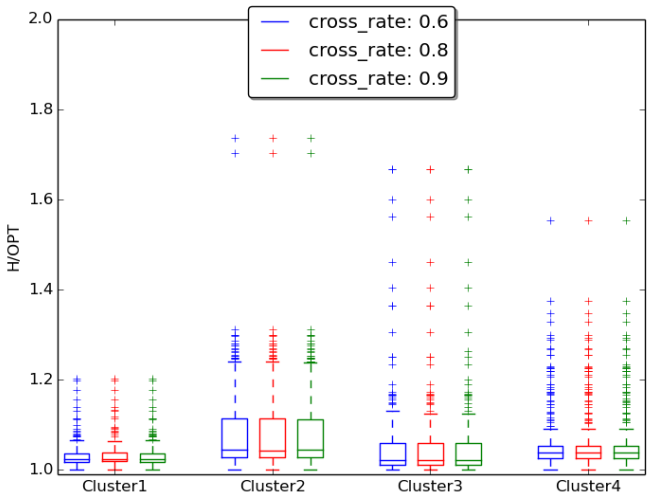


FIGURE 11.4: *The distribution of results returned during a sensitivity analysis in respect of the crossover rate adopted in the best performing Hybrid GA implementations of the first experimental study, namely a hybrid GA implementation employing elitism replacement combined with PMX crossover and SUS selection for the benchmark data in Cluster 1, and a hybrid GA implementation employing elitism replacement together with PMX crossover and tournament selection for the SPP instances in Clusters 2 to 4. The ratio H/OPT represents the packing accuracy, which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT.*

This statistical analysis indicates that the performance of the hybrid GA is not sensitive to the value of the crossover rate employed in the algorithmic implementation. The use of the elitism strategy as a replacement operator in the hybrid GA implementation may be the reason for this relative invariance. The crossover operator obviously creates larger changes in the packing solutions of a population during the genetic algorithmic search, but at some stage of the search, the algorithm may return the best packing solution which would be kept until the end of the procedure.

11.3 Selection of the best Hybrid SA Implementation

Various parameter combinations are also tested in respect of the hybrid SA algorithmic implementation in order to select the best performing parameter value combination for use during the algorithmic comparative study in the next chapter. As was the case for hybrid GA implementations described in §11.2, the SA parameter optimisation experiments also consist of two phases, whereby two sets of parameters are evaluated separately. The set of parameters considered during the first phase are the initial temperature parameter, the cooling parameter, and the epoch parameter. During the second phase, the best combinations of parameter values obtained during the first phase are adopted as constant values, while the fitness function parameter is varied.

Details of the evaluation carried out in terms of these parameters, as well as the respective comparative results returned, are presented in this section. The CH algorithm described in §3.5

is again implemented as decoding routine for the hybrid SA algorithmic incarnations in this preliminary study, because of its superior performance (as documented in Chapter 8).

11.3.1 Experimental Design

As mentioned above, the experimental study involves two types of testing. During the first stage of testing, the same parameter setting analysis as for the two improved algorithms in §9.3 is performed. That is, the first analysis consists of eight experimentations, whereby the hybrid SA combined with the CH algorithm is implemented in different incarnations based on two different values of the initial temperature parameter, two values of the cooling parameter, and two different epoch management strategies. The best output obtained during the first analysis is further subjected to a sensitivity analysis with respect to the fitness function during the second stage of testing.

First experimental study

The hybrid SA is implemented in eight different incarnations based on two different values of the initial temperature T_0 (29.79, 92.54), two values of the cooling parameter β (0.93, 0.95), and two different epoch management strategies (an epoch is terminated once a fixed number N of iterations have been executed or when a total number $\frac{N}{2}$ of successful moves have been attempted, where N denotes the problem instance dimension). These parameter values were selected based on the argument provided in §9.3. A summary of the eight algorithmic incarnations, together with the corresponding parameter values, is provided in Table 11.6.

	$T_0 = 92.54$		$T_0 = 29.79$	
	$\beta = 0.93$	$\beta = 0.95$	$\beta = 0.93$	$\beta = 0.95$
Epoch length: N moves	*HSA1*	*HSA3*	*HSA5*	*HSA7*
Epoch length: $\frac{N}{2}$ successful moves	*HSA2*	*HSA4*	*HSA6*	*HSA8*

TABLE 11.6: *Eight different incarnations of the hybrid SA algorithm combined with the CH algorithm as decoding method, based on different values of the algorithmic parameters, during the parameter evaluation experiment. The parameter T_0 represents the initial temperature, while the other two parameters, β and N, denote the cooling parameter and the problem instance dimension (the number of items involved in a given instance), respectively. The table entries are the names assigned to the experiments.*

The eight hybrid SA implementations are applied to the clustered benchmark data of Chapter 7. These implementations are each executed five times, terminating a run when 5 000 iterations have been executed. A time limit of 60 seconds is additionally imposed for large instances that contain more than 5 000 items. The best solution returned is recorded in each case.

Second experimental study

The second experimental study consists of studying the effect of changing the fitness function employed during the search on the performance of the algorithm in respect of the clustered benchmark instances. The best performing parameter values obtained during the first experimental study are implemented using both the packing height and the area of contiguous remainder as fitness functions, as described in §11.2.1.

11.3.2 Computational Results

The numerical results obtained when following the experimental design described in §11.3.1 are presented in this section. First, the results returned by the eight hybrid SA implementations during the first experimental study are reported. Thereafter, the results returned during the second experimental study, as described in §11.3.1, are discussed. The results are again reported in the form of boxplots and a significance level of $\alpha = 0.05$ is adopted for all the results presented here.

First experimental study results

Boxplots of the results obtained by the eight algorithmic implementations of Table 11.6, when applied to the clustered benchmark instances of Chapter 7, are shown in Figure 11.5. Applying the Friedman test to the results obtained for all the clusters yields a p-value less than 1×10^{-15} in each case, indicating that there are statistically significant differences between the solutions achieved by some of the algorithms at a 5% level of significance. Upon comparing the eight boxplots associated with each cluster, it is clear that a larger value of the initial temperature causes a shift of the results further away from the optimum. More precisely, the four implementations HSA1, HSA2, HSA3, and HSA4 achieved larger mean packing heights than did the other four implementations, independently of the benchmark clusters. This result indicates that a relatively small value of the initial temperature is more suitable for the implementation of the hybrid SA algorithm.

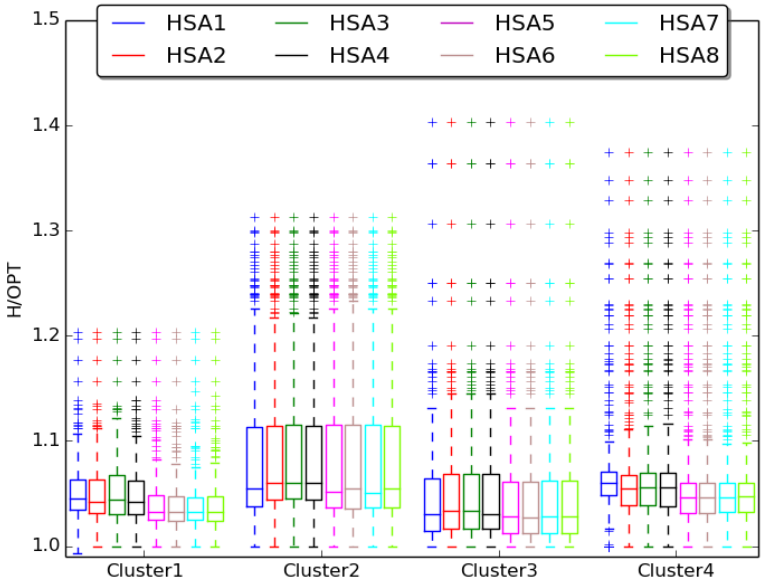


FIGURE 11.5: *The distribution of results returned by the eight Hybrid SA implementations in Table 11.6 with the CH algorithm as decoding procedure, as described in §11.3.1, when applied to the clustered benchmark data. The ratio H/OPT represents the packing accuracy, which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT.*

The p-values returned by the Nemenyi test performed in respect of the eight sets of solutions for the first cluster of benchmark instances in Table 11.7 indicate that there are statistical differences between the solutions achieved by the implementations at a 5% level of significance. The HSA6 implementation performed the best, achieving the smallest mean rank. The HSA7 and

p-values of the Nemenyi rank test: Hybrid SA combined with the CH algorithm (Cluster 1)								
Algorithm	HSA1	HSA2	HSA3	HSA4	HSA5	HSA6	HSA7	HSA8
HSA2	0.998	—						
HSA3	0.0065	0.0504	—					
HSA4	0.0879	0.0132	2.7×10^{-9}	—				
HSA5	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—			
HSA6	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	0.00026	—		
HSA7	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	8.4×10^{-14}	9.4×10^{-5}	1.1×10^{-13}	—	
HSA8	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	7.4×10^{-14}	0.0001	5.4×10^{-14}	1.00	—
Mean (Rank)	1.0592 (4)	1.0578 (4)	1.0585 (4)	1.0573 (4)	1.0489 (3)	1.0484 (1)	1.0487 (2)	1.0486 (2)

TABLE 11.7: *Comparison of the results returned by the eight hybrid SA implementations in Table 11.6 combined with the CH algorithm as decoding procedure in respect of the first cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

p-values of the Nemenyi rank test: Hybrid SA combined with the CH algorithm (Cluster 2)								
Algorithm	HSA1	HSA2	HSA3	HSA4	HSA5	HSA6	HSA7	HSA8
HSA2	6.5×10^{-9}	—						
HSA3	7.7×10^{-14}	—						
HSA4	9×10^{-8}	$< 1 \times 10^{-15}$	0.24	—				
HSA5	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—			
HSA6	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	8.7×10^{-7}	—		
HSA7	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	7.6×10^{-14}	8.8×10^{-7}	—	
HSA8	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	8.5×10^{-14}	4.6×10^{-12}	2.5×10^{6}	7.6×10^{-14}	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.0893 (5)	1.0927 (6)	1.0934 (7)	1.0928 (7)	1.0878 (3)	1.0878 (2)	1.0876 (1)	1.0881 (4)

TABLE 11.8: *Comparison of the results returned by the eight hybrid SA implementations in Table 11.6 combined with the CH algorithm as decoding procedure in respect of the second cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

p-values of the Nemenyi rank test: Hybrid SA combined with the CH algorithm (Cluster 3)								
Algorithm	HSA1	HSA2	HSA3	HSA4	HSA5	HSA6	HSA7	HSA8
HSA2	9.4×10^{-14}	—						
HSA3	6.9×10^{-7}	0.129	—					
HSA4	2.1×10^{-13}	0.999	0.295	—				
HSA5	2.5×10^{-14}	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—			
HSA6	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	0.00351	—		
HSA7	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	7.4×10^{-14}	5.9×10^{-5}	—	
HSA8	1.7×10^{-9}	$< 1 \times 10^{-15}$	8.9×10^{-14}	1.2×10^{-10}	0.0001	8×10^{-14}	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.1024 (5)	1.1042 (6)	1.1044 (6)	1.1040 (6)	1.1005 (3)	1.1003 (2)	1.1003 (1)	1.1006 (4)

TABLE 11.9: *Comparison of the results returned by the eight hybrid SA implementations in Table 11.6 combined with the CH algorithm as decoding procedure in respect of the third cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

p-values of the Nemenyi rank test: Hybrid SA combined with the CH algorithm (Cluster 4)								
Algorithm	HSA1	HSA2	HSA3	HSA4	HSA5	HSA6	HSA7	HSA8
HSA2	1.3×10^{-12}	—						
HSA3	9.1×10^{-14}	4.3×10^{-5}	—					
HSA4	3×10^{-5}	0.71	2.8×10^{-9}	—				
HSA5	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—			
HSA6	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	1.2×10^{-5}	—		
HSA7	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	5.5×10^{-9}	3.1×10^{-8}	—	
HSA8	7.5×10^{-14}	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	1.2×10^{-14}	0.97	$< 1 \times 10^{-15}$	2.8×10^{-6}	—
Mean (Rank)	1.0779 (6)	1.0724 (4)	1.0732 (5)	1.0727 (4)	1.0651 (1)	1.0653 (2)	1.0655 (3)	1.0652 (1)

TABLE 11.10: *Comparison of the results returned by the eight hybrid SA implementations in Table 11.6 combined with the CH algorithm as decoding procedure in respect of the fourth cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the implementations, with their ranks shown in parentheses (a rank of 1 indicates that the implementation achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

HSA8 implementations both performed the second most effectively for this cluster (as they were statistically similar in performance with a p-value of 1.0), followed by the HSA5 implementation. The other four implementations performed relatively poorly, achieving packing heights that are relatively far from the optimum height (yielding an optimality gap of approximately 6%), as shown in the table. According to this statistical analysis, the hybrid SA should preferably be implemented using an initial temperature value of 29.79, a cooling parameter value of 0.93, and a total number of $\frac{N}{2}$ successful moves as an epoch termination trigger, for the Cluster 1 benchmark instances.

Performing the same significance test in respect of the results returned by the eight algorithmic implementations for the second cluster of benchmark instances indicates that there is no significant difference between the results returned by the HSA3 and HSA4 implementations, but that there are significant differences between the results returned by all other pairs of implementations (see Table 11.8). The group of implementations employing an initial temperature value of 29.79 outperformed the other four for this cluster as well. Of the four implementations in this group, the HSA7 implementation achieved the best mean rank, while the HSA8 implementation achieved the worst mean rank. Accordingly, the HSA7 implementation should be selected as hybrid SA implementation for the SPP instances in Cluster 2.

The comparison of the eight algorithmic implementations in respect of the third cluster yielded relatively similar results as for the second cluster. The HSA7 implementation returned the most favourable results, significantly outperforming all other implementations at a 5% level of significance, as shown in Table 11.9. This suggests that the hybrid SA algorithm should be implemented using an initial temperature value of 29.79, a cooling parameter value of 0.95, and a fixed number of N iterations as an epoch termination trigger, for the Cluster 2 data. The group of implementations employing an initial temperature value of 92.54 again perform relatively poorly for this cluster.

The order of the relative performances of the eight algorithmic implementations changed slightly in respect of the fourth data cluster. The HSA5 and HSA8 implementations did not perform statistically differently at a 5% level of significance, both achieving the smallest mean packing height solutions, as shown in Table 11.10. The HSA6 implementation achieved the second smallest packing height solution, and was followed by the HSA7 implementation. Based on this result, the HSA5 implementation is selected arbitrarily as the hybrid SA implementation in this study for the Cluster 4 benchmark instances.

In summary, the performance of the hybrid SA algorithm is sensitive with respect to the value of the initial temperature implemented. A relatively small value of the initial temperature is statistically suggested for adoption in the hybrid SA algorithm. Furthermore, a cooling parameter value of 0.93 may be implemented in the hybrid SA for the benchmark instances in Clusters 1 and 4, whereas a value of 0.95 is preferred for the SPP instances in Clusters 2 and 3. Finally, it is statistically recommended that epochs should be terminated after having performed a fixed number of N iterations, where N denotes the problem dimension, for the benchmark instances in Clusters 1 to 3, while an epoch should be terminated when $\frac{N}{2}$ successful moves have been attempted during the algorithmic search for the Cluster 4 data.

Second experimental study results

Based on the first experimental study results, the HSA6 implementation (a hybrid SA implementation employing an initial temperature value of 29.79, a cooling parameter value of 0.93, and a total number of $\frac{N}{2}$ successful moves as an epoch termination trigger) is the best performing hybrid GA implementation for the SPP instances in Cluster 1, while the HSA7 implementation (a hybrid SA implementation employing an initial temperature value of 29.79, a cooling parameter value of 0.95, and a total number of N iterations as an epoch termination trigger) is the most suitable hybrid SA implementation for the benchmark data in Clusters 2 and 3, and the HSA5 implementation (a hybrid SA implementation employing an initial temperature value of 29.79, a cooling parameter value of 0.93, and a total number of N iterations as an epoch termination trigger) is preferable for Cluster 4 data.

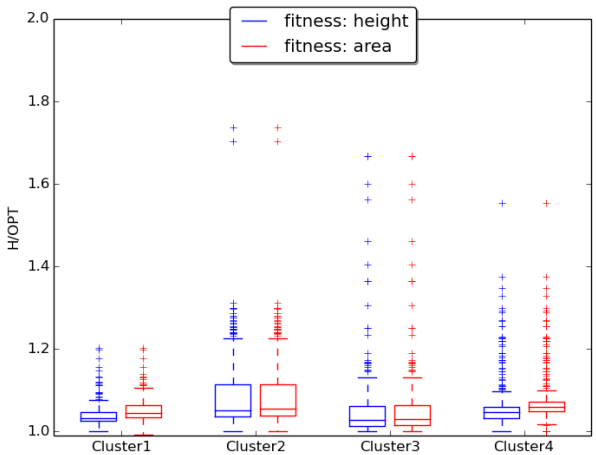


FIGURE 11.6: *The distribution of results returned during a sensitivity analysis in respect of the fitness function employed within the best performing hybrid SA implementations of the first experimental study, namely a hybrid SA implementation employing an initial temperature value of 29.79, a cooling parameter value of 0.93, and a total number of $\frac{N}{2}$ successful moves as an epoch termination trigger for the SPP instances in Cluster 1, a hybrid SA implementation employing an initial temperature value of 29.79, a cooling parameter value of 0.95, and a total number of N iterations as an epoch termination trigger for the benchmark data in Clusters 2 and 3, and a hybrid SA implementation employing an initial temperature value of 29.79, a cooling parameter value of 0.93, and a total number of N iterations as an epoch termination trigger for Cluster 4 data. Here, the parameter N denotes the problem instance dimension. The ratio H/OPT represents the packing accuracy, which is the ratio between the mean strip height H achieved by an implementation and the optimal height (or the appropriate lower bound) OPT. Results returned by the Hybrid SA implementations employing packing height as fitness function are represented by blue boxplots, while the results are represented by red boxplots for the implementation employing area of contiguous remainder as fitness function.*

The aforementioned hybrid SA implementations were considered during the second experimental study, involving application of two different fitness functions to the algorithmic implementations, as described in §11.3.1. Boxplots of the corresponding results, when applied to the various clustered benchmark instances, are shown in Figure 11.6. As was the case for hybrid GA discussed in §11.2.2, a similar trend emerged here with respect to the order of the relative performances of the two algorithmic incarnations for all the clusters. The hybrid SA implementation utilising packing height as fitness function yielded better packing solutions for all the clusters than does the implementation employing the area of contiguous remainder as fitness function. This observation is clear from the boxplots in Figure 11.6, and is supported by applying the Friedman test to the results for all the clusters, yielding p-values less than 1×10^{-15} in each case.

Again, these results are not surprising. As explained for the case of hybrid GA in §11.2.2, the use of the area of contiguous remainder as fitness function in the hybrid SA algorithmic implementation encourages the search process to examine packing solutions with relatively large packing heights, thus leading to the possibility of achieving a relatively poor solution quality at the end of the search. Accordingly, it is advisable to employ packing height as fitness function in the algorithmic implementation of the hybrid SA.

11.4 Chapter Summary

This chapter was devoted to a comparison of different algorithmic incarnations of the strip packing hybrid metaheuristics reviewed in Chapter 4. A limited computational study was conducted with the specific aim of identifying the most effective algorithmic implementation of these approaches. The method of comparative analysis was first described in §11.1. Thereafter, details of the experimental design and the corresponding results were presented in §11.2 for the case of

		Cluster 1	Cluster 2	Cluster 3	Cluster 4
Hybrid GA	Crossover operator	PMX	PMX	PMX	PMX
	Selection operator	SUS	Tournament	Tournament	Tournament
	Replacement strategy	Elitism	Elitism	Elitism	Elitism
	Fitness function	Packing height	Packing height	Packing height	Packing height
	Crossover rate	60%	60%	60%	60%
	Mutation rate	3%	3%	3%	3%
Hybrid SA	Initial temperature	29.79	29.79	29.79	29.79
	Cooling parameter	0.93	0.95	0.95	0.93
	Epoch length	Fixed number N of moves	Fixed number N of moves	Fixed number N of moves	Fixed number $\frac{N}{2}$ of successful moves
	Fitness function	Packing height	Packing height	Packing height	Packing height

TABLE 11.11: *Summary of the recommended algorithmic implementations for both the hybrid GA and hybrid SA algorithms with respect to each cluster of benchmark instances. The parameter N denotes the problem instance dimension.*

the hybrid GA implementations. The computational results achieved in respect of hybrid SA implementations were finally discussed in §11.3.

A summary of the most effective algorithmic implementations for both the hybrid GA and hybrid SA algorithms with respect to each cluster of benchmark instances may be found in Table 11.11. This table indicates which operators and parameter values should be utilised for the algorithmic implementations of these algorithms.

CHAPTER 12

Appraisal of Strip Packing Metaheuristics

Contents

This chapter contains a discussion on the results returned during an experimental study involving a comparison of the relative effectiveness of the most effective implementations of all SPP metaheuristics under investigation in this dissertation, namely the seven SPP solution approaches reviewed in Chapter 4 and the two adapted algorithms proposed in Chapter 8, in respect of the clustered benchmark data of Chapter 7. The comparison is carried out in terms of both solution quality and execution time, and the corresponding results are presented in the form of boxplots, tables of *post hoc* test results, and graphs.

The chapter opens in §12.1 with a presentation of the method of evaluation performed in the above-mentioned comparative study. This is followed by descriptions of the parameter settings selected for each of the metaheuristic implementations in §12.2. Since the algorithmic implementations of the two adapted algorithms, and the two hybrid metaheuristics (hybrid GA and hybrid SA)[1] were presented in Chapters 9 and 11, respectively, only the implementations of the other five algorithms are discussed in the aforementioned section. Thereafter, the comparative study results are reported in §12.3, while a characterisation of the effectiveness of the various

[1]Throughout this chapter, the term *hybrid GA* refers to a combination of a GA with the CH algorithm of §3.5 as decoding heuristic, while *hybrid SA* refers to a combination of the method of SA with the CH algorithm of §3.5 as decoding heuristic.

algorithms is provided in §12.4 for the four benchmark clusters in terms of algorithmic execution time. The chapter closes in §12.5 with a summary of the chapter contents.

12.1 Evaluation of Strip Packing Metaheuristics

As mentioned before, two types of analysis may be performed in comparative studies of the relative effectiveness of metaheuristics. The first type deals with results obtained over several execution runs of metaheuristics in respect of a particular problem instance, whereas results returned by a single algorithmic implementation applied to a set of benchmark problem instances are considered in the second type of analysis. In this chapter, both these types of analysis are performed. Each metaheuristic is repeatedly applied to an SPP benchmark instance because of stochastic elements inherent in the algorithms, thus producing a sample of packing heights for each benchmark instance. The best results returned by each algorithm in respect of a set of benchmark instances are then compared. More precisely, the comparison of the relative effectiveness of the different SPP metaheuristic implementations takes place in respect of the best results returned by each algorithmic implementation applied to every member of the sets of clustered benchmark data described in Chapter 7.

12.2 Strip Packing Metaheuristic Implementations

In this section, detailed descriptions are provided of how the five known metaheuristics reviewed in Chapter 4, namely the SPGAL, the reactive GRASP, the ISA, the SRA, and the IA algorithms, were implemented. The selected parameter settings of these algorithms are declared, and the simulations performed in respect of each algorithm are described. All algorithms were coded in Python using Spyder Version 2.7.6. The algorithms were, furthermore, all executed on an Intel Core i7-4790 CPU running at 3.60 GHz with 8 GB RAM in the Ubuntu 14.04 operating system.

12.2.1 The SPGAL Algorithm

The parameter settings suggested by Bortfeldt [25] were implemented in the SPGAL algorithm for all the four benchmark clusters in this study. The integrated CLP-GA in the SPGAL algorithm was implemented with a population size *npop* of 50. The crossover probability was selected as 67% and the mutation probability as 33%. The number *nrep* of solutions to be reproduced during each iteration, as well as the number *nmerge* of solutions to be generated during the course of a merger mutation per generation, were both taken as 10. The number *ngen* of generations per instance depends on the number of items n involved in the given instance. Its value was taken as follows: $ngen = 1\,000$ if $n \in [1, 60]$, $ngen = 500$ if $n \in [61, 100]$, or $ngen = 100$ otherwise.

As described in §4.2.3, a restriction on the number of permitted layer variants was imposed during the completion of a partial solution in order to reduce the required execution time. This is controlled by the parameters *qℓdp1* and *qℓdp2* during the course of a crossover, and by the percentage *qℓdp3* in the case of mutation. The value of these percentages depends on the number *nptypes* of item types in the given instance. Their respective values are summarised in Table 12.1.

The parameter *nplarge* was taken as 199, *i.e.* a problem instance was reduced in size for the CLP-GA if it contains at least 200 items, while the parameter *npsmall* was taken as 100, *i.e.* the

nptypes lies in	[1,40]	[41,60]	[61,200]	> 200
qℓdp1	100	66	10	5
qℓdp2	100	66	10	5
qℓdp3	33	33	33	33

TABLE 12.1: *Value of the percentages qℓdpi, $i = 1, 2, 3$, implemented in the SPGAL algorithm of §4.2.3 in this study. The parameter nptypes represents the number of item types or congruent items.*

number of items in a reduced instance exceeds 100. The number of runs per SPP instance was taken as 5, where one run was terminated when 1 000 iterations had been carried out or an infeasible solution was obtained. For large instances that contain more than 1 000 items, the stopping criterion was defined as a time limit of 60 seconds.

12.2.2 The Reactive GRASP Algorithm

The best strategies for defining the GRASP algorithm proposed by Alvarez-Valdés *et al.* [5] were implemented for all four benchmark clusters in this dissertation. During the constructive phase of the algorithm, an item was selected for packing from a restricted set of candidates according to a scoring criterion. Each unpacked item i was associated with a score $s_i = \omega_i + (k \times h_i)$, where ω_i and h_i represent the respective width and height of the item, and the parameter k was generated randomly in the interval (0.01, 0.75). An item was selected for packing during each step of the constructive procedure from a set $C = \{j \mid s_j \geq s_{\min} + \gamma(s_{\max} - s_{\min})\}$, where $s_{\max} = \max\{s_b$, for all items b in the given instance$\}$, $s_{\min} = \min\{s_b$, for all items b in the given instance$\}$, and where γ was determined according to the reactive GRASP described in §4.2.4. The value of the parameter λ employed in the reactive GRASP was fixed at 10 as in [5].

Untidy arrangements caused by the randomised constructive procedure were corrected during the improvement phase of the algorithm. The last $k\%$ items of the solution packed in the constructive phase (for instance, the last 20%) were removed and the resulting empty space was filled by means of the deterministic constructive algorithm described in §4.2.4. The percentage was chosen so as to guarantee that the packing height returned by the process is strictly smaller than the original packing height. If this was not the case, more items were removed from the initial solution until this condition was indeed satisfied.

For the sake of comparison, the algorithm was executed five times for each SPP instance and the best solution returned was recorded. A time limit of 60 seconds was additionally imposed for large instances that contain more than 1 000 items.

12.2.3 The Two-stage Intelligent Search Algorithm

The geometric annealing schedule employed by Leung *et al.* [116] was also implemented in the two-stage ISA algorithm for all four benchmark clusters in this study. The cooling rate was taken as 0.93. The initial value of the temperature was determined as 0.5. The temperature was furthermore held constant for a fixed number N of iterations, with N representing the problem instance dimension.

The algorithm was again executed five times for each SPP instance, where one run was terminated when 5 000 search iterations had been carried out. A time limit of 60 seconds was additionally imposed for large instances that contain more than 1 000 items. The best solution returned was recorded.

12.2.4 The Simple Randomised Algorithm

Since no parameter settings are required for the implementation of the SRA algorithm, the algorithm was implemented as-is and was also executed five times for each SPP instance, after which the best solution returned was recorded. The stopping criterion was taken as a time limit of 60 seconds for large instances that contain more than 1 000 items, while the search was terminated when 5 000 search iterations had been carried out for small and medium-sized benchmark instances.

12.2.5 The Improved Algorithm

No parameter settings are required for the implementation of the IA algorithm. Hence it was implemented following the pseudocode listed in §4.2.7. The algorithm was executed five times for each SPP benchmark, and a run was terminated when 5 000 search iterations had been carried out for small and medium-sized benchmark instances, while a time limit of 60 seconds was imposed for large instances that contain more than 1 000 items. The best solution returned was recorded in each case.

12.3 Comparison of Strip Packing Metaheuristic Results

The relative performances of the nine metaheuristics, in terms of both solution quality and execution time, were compared with one another in respect of the four benchmark clusters of Chapter 7. The comparison took place in respect of the best output of each algorithm, obtained from the algorithmic implementations described above. The results are presented in this section and are interpreted by means of boxplots and tables indicating whether or not differences exist between the performances of each pair of algorithms in respect of each benchmark cluster at a 5% level of significance.

12.3.1 Comparison in terms of Solution Quality

A comparison of the nine metaheuristic algorithms shows that the newly proposed IAm algorithm competes favourably with the known algorithms, while the SPSAL algorithm outperforms some of the existing metaheuristics. Moreover, the best hybrid GA algorithmic implementation, obtained from the experimental design conducted in Chapter 11, yields superior results in respect of some SPP instances. Boxplots of the performances of these algorithms are shown in Figure 12.1, and the presence of significant differences between the results returned by the various algorithms is elucidated in the Nemenyi rank test results in Tables 12.2–12.5.

For each cluster, the relative performances of the three algorithms, namely the IAm, the ISA, and the hybrid GA, do not differ from each other statistically at a 5% level of significance. The SPSAL, the SPGAL, and the GRASP algorithms, on the other hand, packed the majority of the benchmark instances to a larger relative packing height than did the other six algorithms. Finally, the mean rank of some of the algorithms changed depending on the benchmark clusters.

In respect of the first cluster of data, the IAm, the ISA, the SRA, the hybrid GA, and the hybrid SA performed statistically similar in performance, all achieving the smallest mean packing height, as shown in Table 12.2. The IA algorithm was the next best performing algorithm in this case, followed by the GRASP algorithm. The SPSAL and SPGAL algorithms were the worst performing algorithms for this cluster.

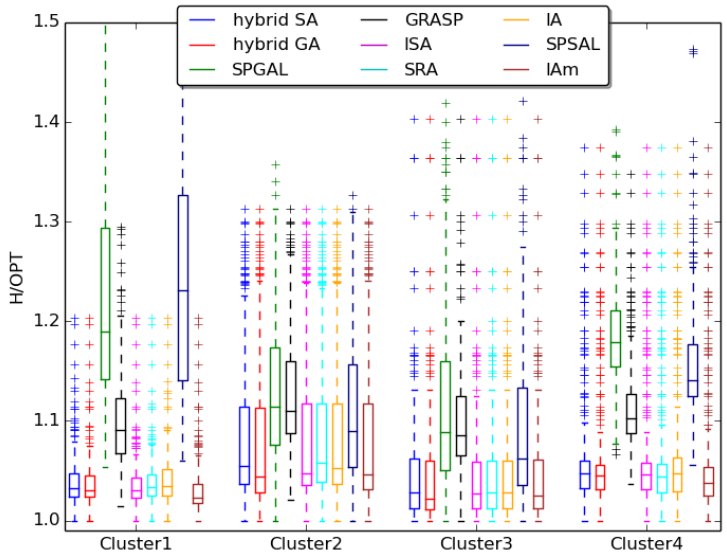


FIGURE 12.1: *The distribution of results returned by all the SPP metaheuristics under investigation in this dissertation, when applied to the clustered benchmark data of Chapter 7. These metaheuristics are the SPGAL algorithm of Bortfeldt [25], the GRASP algorithm of Alvarez-Valdés et al. [5], the ISA algorithm of Leung et al. [116], the SRA of Yang et al. [162], the IA of Wei et al. [158], the hybrid GA and hybrid SA algorithms of Chapter 11, and the two newly proposed metaheuristics (the IAm algorithm and the SPSAL algorithm) of Chapter 8. The ratio H/OPT represents the packing accuracy, which is the ratio between the mean strip height H achieved by an algorithm and the optimal height (or the lower bound) OPT.*

p-values of the Nemenyi rank test: All metaheuristic algorithms (Cluster 1)									
Algorithm	hybrid SA	hybrid GA	SPGAL	GRASP	ISA	SRA	IA	SPSAL	IAm
hybrid GA	0.148	—							
SPGAL	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—						
GRASP	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	7.8×10^{-5}	—					
ISA	0.972	0.813	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—				
SRA	1.00	0.074	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	0.905	—			
IA	0.999	0.019	$< 1 \times 10^{-15}$	1.6×10^{-14}	0.671	1.00	—		
SPSAL	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	1.00	7.2×10^{-5}	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—	
IAm	0.182	1.00	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	0.984	1.00	0.01	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.0486 (1)	1.0407 (1)	1.2345 (4)	1.1113 (3)	1.0448 (1)	1.0466 (1)	1.05 (2)	1.2461 (4)	1.04 (1)

TABLE 12.2: *Comparison of the results returned by all the SPP metaheuristics considered in this dissertation, namely the seven metaheuristics described in Chapter 4 and the two newly proposed algorithms presented in Chapter 8, in respect of the first cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the algorithms, with their ranks shown in parentheses (a rank of 1 indicates that the algorithm achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

In respect of the second cluster of data, the order of the relative performance of the algorithms changed slightly. The IA algorithm yielded a relatively high solution quality, competing with the IAm, the ISA, the hybrid GA and the hybrid SA algorithms. The SRA algorithm became the second best performing algorithm in this case. Moreover, the performance of the SPSAL algorithm improved in respect of this cluster, significantly outperforming the SPGAL and GRASP algorithms (see Table 12.3).

The comparison of the nine algorithms in respect of the third benchmark cluster yielded relatively similar results as for the second benchmark cluster. The results returned by the IAm, the ISA, the SRA, the IA, the hybrid GA and the hybrid SA algorithms do not differ statistically at a 5% level of significance, outperforming the other three algorithms. The SPSAL algorithm was ranked the second most effectively in respect of this cluster, followed by the SPGAL and GRASP algorithms (see Table 12.4).

p-values of the Nemenyi rank test: All metaheuristic algorithms (Cluster 2)									
Algorithm	hybrid SA	hybrid GA	SPGAL	GRASP	ISA	SRA	IA	SPSAL	IAm
hybrid GA	0.614	—							
SPGAL	6.9×10^{-14}	$< 1 \times 10^{-15}$	—						
GRASP	7.7×10^{-14}	$< 1 \times 10^{-15}$	0.999	—					
ISA	1.00	0.804	8.9×10^{-14}	9.6×10^{-14}	—				
SRA	0.047	0.045	7.5×10^{-14}	6.1×10^{-14}	0.045	—			
IA	0.105	7.7×10^{-5}	2.4×10^{-11}	1.1×10^{-9}	0.045	0.328	—		
SPSAL	0.0019	1.2×10^{-7}	6.7×10^{-8}	1.6×10^{-6}	0.0005	0.0137	0.961	—	
IAm	1.00	0.676	9.3×10^{-14}	7.0×10^{-14}	1.00	0.045	0.082	0.0013	—
Mean (Rank)	1.075 (1)	1.07 (1)	1.122 (4)	1.119 (4)	1.074 (1)	1.08 (2)	1.078 (1)	1.103 (3)	1.073 (1)

TABLE 12.3: *Comparison of the results returned by all the SPP metaheuristics considered in this dissertation, namely the seven metaheuristics described in Chapter 4 and the two newly proposed algorithms presented in Chapter 8, in respect of the second cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the algorithms, with their ranks shown in parentheses (a rank of 1 indicates that the algorithm achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

p-values of the Nemenyi rank test: All metaheuristic algorithms (Cluster 3)									
Algorithm	hybrid SA	hybrid GA	SPGAL	GRASP	ISA	SRA	IA	SPSAL	IAm
hybrid GA	0.998	—							
SPGAL	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—						
GRASP	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	1.00	—					
ISA	0.66	0.974	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—				
SRA	1.00	1.00	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	0.931	—			
IA	1.00	0.998	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	0.643	1.00	—		
SPSAL	8.5×10^{-14}	9.1×10^{-14}	0.028	0.047	2.0×10^{-14}	8.0×10^{-14}	8.6×10^{-14}	—	
IAm	0.969	1.00	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	0.99	1.00	0.965	8.9×10^{-14}	—
Mean (Rank)	1.1006 (1)	1.0984 (1)	1.1646 (3)	1.1533 (3)	1.0995 (1)	1.1002 (1)	1.1008 (1)	1.1450 (2)	1.0997 (1)

TABLE 12.4: *Comparison of the results returned by all the SPP metaheuristics considered in this dissertation, namely the seven metaheuristics described in Chapter 4 and the two newly proposed algorithms presented in Chapter 8, in respect of the third cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the algorithms, with their ranks shown in parentheses (a rank of 1 indicates that the algorithm achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

The results obtained in respect of the fourth cluster of data were different to those of the previous clusters. In this case, the hybrid SA algorithm was ranked the second most effectively, significantly outperformed by the five algorithms, namely the IAm, the ISA, the SRA, the IA and the hybrid GA algorithms. The results returned by these five algorithms were statistically similar at a 5% level of significance, all achieving the smallest mean packing height for the majority of instances in this cluster. The SPSAL and SPGAL algorithms were ranked the last in respect of this cluster, significantly outperformed by the GRASP algorithm (see Table 12.5).

p-values of the Nemenyi rank test: All metaheuristic algorithms (Cluster 4)									
Algorithm	hybrid SA	hybrid GA	SPGAL	GRASP	ISA	SRA	IA	SPSAL	IAm
hybrid GA	0.045	—							
SPGAL	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—						
GRASP	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	6.5×10^{-11}	—					
ISA	0.031	1.00	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—				
SRA	0.04	0.993	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	0.982	—			
IA	0.03	0.995	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	0.998	0.716	—		
SPSAL	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	0.2416	0.0001	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	—	
IAm	0.047	1.00	$< 1 \times 10^{-15}$	$< 1 \times 10^{-15}$	0.99	0.99	0.975	$< 1 \times 10^{-15}$	—
Mean (Rank)	1.081 (2)	1.0653 (1)	1.1916 (4)	1.1244 (3)	1.0671 (1)	1.0659 (1)	1.0685 (1)	1.1654 (4)	1.0644 (1)

TABLE 12.5: *Comparison of the results returned by all the SPP metaheuristics considered in this dissertation, namely the seven metaheuristics described in Chapter 4 and the two newly proposed algorithms presented in Chapter 8, in respect of the fourth cluster of benchmark instances. Red entries indicate statistical differences at a 5% level of significance. The row labelled 'Mean (Rank)' contains the mean performance ratio achieved by the algorithms, with their ranks shown in parentheses (a rank of 1 indicates that the algorithm achieved the smallest mean packing height for all instances in the cluster of benchmark instances).*

12.3.2 Comparison in terms of Execution Time

The average run time of all nine SPP metaheuristics increased exponentially as a function of the problem instance size, as is clear from studying the natural logarithms of these execution times, shown in Figure 12.2. There is, however, a significant difference between the nine algorithms in terms of time taken to solve SPP instances, especially for large problem instances involving more than 500 items (applying the Friedman test yielded a p-value smaller than 1×10^{-15} in this respect).

The hybrid SA algorithm was the fastest algorithm, requiring 39 minutes to solve an SPP instance containing 500 items, while the IA algorithm appeared to be the slowest algorithm, requiring approximately 10 hours to solve a similarly sized instance (see Table 12.6). The IAm algorithm is the second most efficient algorithm in terms of computation time, followed by the SPSAL and SPGAL algorithms. Although the hybrid GA algorithm achieves better results for some SPP instances, it requires longer computation times to solve the majority of the problem instances than do the other algorithms, except for the GRASP algorithm. The ISA and SRA algorithms also require long run times to solve most of the problem instances, and these run times become extremely long for large problem instances due to the algorithms' higher computational complexities.

12.3.3 Discussion

Considering the results reported in §12.3.1 and §12.3.2, it is clear that the IAm algorithm performs well on average, outperforming all other metaheuristics in respect of Clusters 1 and 4 of the benchmark data. The hybrid GA algorithm is, however, ranked the best performing algorithm in respect of the benchmark instances in Clusters 2 and 3, but its performance does not differ significantly from the performance of the IAm algorithm in those cases (see Tables 12.3–12.4). Furthermore, the hybrid GA algorithm is significantly slower than the IAm algorithm (see Figure 12.2 and Table 12.6).

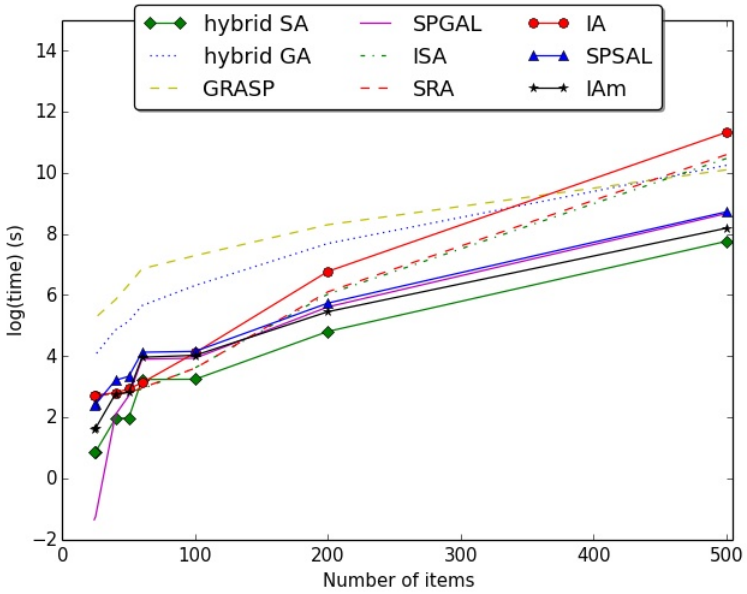


FIGURE 12.2: *Natural logarithm of the execution times of all the SPP metaheuristics under investigation in this dissertation as a function of the number of items per SPP instance. These metaheuristics are the SPGAL algorithm of Bortfeldt [25], the GRASP algorithm of Alvarez-Valdés et al. [5], the ISA algorithm of Leung et al. [116], the SRA of Yang et al. [162], the IA of Wei et al. [158], the hybrid GA and hybrid SA algorithms of Chapter 11, and the two newly proposed metaheuristics (the IAm algorithm and the SPSAL algorithm) of Chapter 8.*

	$N = 20$	$N = 50$	$N = 100$	$N = 200$	$N = 500$
hybrid SA	0.04	0.42	0.43	2.04	39.03
hybrid GA	0.95	4.76	9.19	36.33	468.86
SPGAL	0.02	0.82	0.85	4.56	96.81
GRASP	3.18	15.86	24.44	67.14	404.25
ISA	0.25	0.31	0.62	6.93	586.33
SRA	0.25	0.32	0.61	7.47	663.92
IA	0.25	0.38	1.01	14.57	1 379.1
SPSAL	0.18	1.03	1.06	5.17	102.24
IAm	0.08	0.87	0.93	3.9	59.93

TABLE 12.6: *Average computation times (in minutes) of the nine SPP metaheuristics considered in this dissertation per SPP instance run, involving N items. These metaheuristics are the SPGAL algorithm of Bortfeldt [25], the GRASP algorithm of Alvarez-Valdés et al. [5], the ISA algorithm of Leung et al. [116], the SRA of Yang et al. [162], the IA of Wei et al. [158], the hybrid GA and hybrid SA algorithms of Chapter 11, and the two newly proposed metaheuristics (the IAm algorithm and the SPSAL algorithm) of Chapter 8.*

The use of a multi-start SA technique together with an evaluation rule based on eight different cases for selecting appropriate items to pack during the decoding process is the principal advantage of the IAm algorithm in terms of generating superior results. The multi-start SA technique can explore the search space more effectively and concentrate on promising regions. Moreover, the embedded evaluation strategy enables the algorithm to identify the best item to pack during the packing process, allowing it to generate dense packing layouts.

As opposed to the multi-start SA method, which operates only on a single solution at a time to generate the best packing order, the GA technique employed in the hybrid GA algorithm explores a population of solutions simultaneously for this purpose. This allows for exploration

of the search space in parallel in the case of the hybrid GA algorithm. Furthermore, the crossover operator employed in the GA to manipulate the current best solutions creates larger changes in the packing order of solutions than does the manipulation technique employed in the multi-start SA method. The recombination method in the GA also guarantees that the most successful solutions are utilised during the following generation. These properties allow the hybrid GA algorithm to generate relatively superior results for certain SPP problem instances, but the relative performance gain, in terms of both solution quality and execution time (especially for large problem instances) justifies application of the IAm algorithm to the 2D SPP.

The results obtained in §12.3.1 also suggest that the IAm algorithm is a significant improvement over the IA algorithm in respect of the first cluster of benchmark data (the mean packing height is 1% better for the new algorithm than for the original). Improvements are also observed with respect to the mean packing height of solutions achieved by the IAm algorithm over those obtained by the IA algorithm in respect of the other three benchmark data clusters (see Tables 12.3–12.5). In addition, the IAm algorithm produces high-quality solutions within reasonable time frames (it was found to be significantly faster than the IA algorithm, see Table 12.6).

These results confirm the claim in §8.1 that the IAm is an improvement of the IA algorithm in terms of both solution quality and execution time. The method of SA in the new algorithm is capable of leading the search into promising areas of the search space, improving on the solution quality of the current solution more effectively than the randomised improvement procedure of the original algorithm. Moreover, exclusion of the original local improvement method in the IAm algorithm enhances the search capability of the integrated SA method on one hand, and reduces the computational time required to execute the entire algorithm on the other hand.

The IAm algorithm not only achieves high-quality packing solutions within a practicable execution time, but its main advantage lies in the relative simplicity of its implementation. It is a simple hybrid approach, whereby the SA search technique is combined with a construction heuristic procedure. The representation of a packing problem in the form of a permutation facilitates the use of the well-known swap manipulation technique during the SA search process and also decreases the effort of the embedded heuristic procedure during the decoding process. No other operators are required in the approach.

The comparative study results reported in §12.3.1 further demonstrate that the SPSAL algorithm is a significant improvement on the SPGAL algorithm in respect of the benchmark instances of Clusters 2 and 3 (with a mean packing height difference of approximately 2%). The relative performances of the two algorithms do not differ significantly in respect of Clusters 1 and 4 benchmark data, although the SPSAL algorithm achieved better mean packing solutions in respect of the first data cluster (see Tables 12.2 and 12.5).

These results corroborate the claim in §8.2 that the SPSAL algorithm, which does not employ any encoding of solutions, is suitable for application to the 2D SPP. It is an improvement on the SPGAL algorithm in respect of all benchmark clusters, except for the first data cluster. The SPSAL algorithm is relatively slower than the SPGAL algorithm, although the performance achieved in terms of solution quality validates its application to the 2D SPP.

It is also noticeable that the mean performance ratio ranks of the various algorithms change, depending on the SPP benchmark clusters. Some algorithms performed relatively better than others with respect to specific benchmark clusters, whereas the same algorithms performed worse for other benchmark instances. For example, the SPSAL algorithm significantly outperformed the GRASP algorithm in respect of Clusters 2 and 3 of the benchmark data, while the opposite was observed in respect of the benchmark instances of Clusters 1 and 4, as shown in Tables 12.2–12.5. Similarly, the IA algorithm outperformed the SRA algorithm in respect of the Cluster 2

benchmark data, while the SRA algorithm achieved better solutions than the IA algorithm in respect of the first benchmark cluster. The characteristics of the benchmark instances may thus affect the relative performances of the various algorithms and need to be taken into consideration during a comparative algorithmic study.

12.3.4 Result Differences

A number of differences are discernible between the results reported in §12.3.1 and those reported in the literature. In [158], the IA algorithm was reported to outperform previously published metaheuristics, including the ISA and SRA algorithms, in respect of the majority of SPP instances. In contrast to this result, it was found in this dissertation that the IA algorithm was outperformed by the latter two algorithms in respect of the first benchmark cluster and the performances of the three algorithms did not differ significantly from each other in respect of the second, third, and fourth benchmark clusters. The reason for this contradiction may be the use of a different experimental setup. A time limit of 60 seconds was imposed during the execution of the stated three algorithms in the aforementioned paper, causing each algorithm to terminate early for the majority of the problem instances considered. The SA technique employed in the ISA algorithm, as well as the randomised improvement procedures implemented in the SRA and IA algorithms, therefore eventually yields almost no improvement on the solutions. Consequently, the IA algorithm seems to provide favourable results since the integrated constructive algorithm employed is more powerful than those employed in the ISA and SRA algorithms. In this study, however, each algorithm was executed without a time restriction for the majority of the benchmark instances. The non-execution of the improvement process implemented in each algorithm was thus largely prevented in order to avoid biased conclusions.

In the same vein, the computational results in [162] show that the SRA algorithm outperforms the state-of-the-art GRASP and ISA algorithms for most SPP benchmark instances. The results obtained in §12.3.1, however, only confirm the claim that the SRA algorithm is superior to the GRASP algorithm, while the SRA algorithm was shown to be inferior to the ISA algorithm for the majority of benchmark instances under consideration. Again the reason for this contradiction may be the use of a different execution experimental setup, as a time limit of 60 seconds was also imposed during the execution of these three algorithms in the stated paper. The algorithms were thus terminated prematurely, which may lead to biased results.

Noteworthy findings of this dissertation also include the fact that a hybrid GA is more competitive than a hybrid SA procedure, contradicting the results reported in [31, 91]. The results returned by a hybrid GA combined with an algorithm in the class of bottom-left heuristics as decoding procedure was shown to be inferior to those returned by a hybrid SA combined with the same heuristic in the aforementioned papers. The opposite result was found in this study: A hybrid GA combined with any of the heuristics of Chapter 3 performed relatively better than a hybrid SA combined with the same heuristics. A plausible explanation for this inconsistency may be the use of different sets of SPP packing problem instances as test beds in the comparative studies. A wide range of benchmark instances was employed in this dissertation in order to compare the relative performances of different algorithms, while the comparison took place in respect of much smaller data sets in the two papers mentioned above. The characteristics of the instances employed may affect the performance of the algorithms.

It is, of course, important to incorporate the most suitable operators and parameter values in hybrid metaheuristic implementations in order to obtain high-quality solutions by means of these solution approaches and also to avoid biased conclusions. In fact, the results reported in §11.2.2 and §11.3.2 suggest that some of the hybrid GA implementations returned solutions

of lower quality than did certain hybrid SA implementations in respect of all four benchmark clusters. The opposite conclusion is, however, reached in certain cases where the hyrbid GA was implemented in conjunction with appropriate operators. A similar conclusion as in [31, 91] might thus be drawn if an inappropriate combination of operators and/or parameter values were to be employed in the hybrid GA and hybrid SA algorithms.

12.4 Characterisation of Strip Packing Algorithms

According to comparative results reported in this dissertation, algorithms in the class of SPP metaheuristic solution approaches predominantly outperform heuristic packing techniques in terms of solution quality. The heuristic packing algorithms are, however, considerably faster in terms of execution time, but yield results that are significantly worse than those obtained by the metaheuristics. For industrial-sized problems, the question which solution approach to adopt is therefore a trade-off between material cost and decision time. In this study, the IAm algorithm achieved the best packing solutions for benchmark instances in Clusters 1 and 4, while the hybrid GA yielded favourable results in respect of Clusters 2 and 3 data. The hybrid GA was, however, found to be very slow in terms of execution time, whereas the IAm algorithm is quick and also yields superior results in respect of the latter clusters. Hence, if the time available for solving a packing problem instance is limited, use of the IAm algorithm is recommended. For a very small amount of time available, the heuristics CH and IHR algorithms are better able to meet this criterion.

A summary of which methods are recommended for large instances in the different benchmark clusters under limited execution time is provided in Table 12.7. This table indicates which method to select as a packing algorithm when solving instances in each cluster within a specific execution time budget.

Time	$< 1h$	$\pm 2h$	$> 5h$
Cluster 1	CHDP	IAm	IAm
Cluster 2	IHRDH	IAm	Hybrid GA
Cluster 3	CHDP	IAm	Hybrid GA
Cluster 4	CHDP	IAm	IAm

TABLE 12.7: *Summary of the best performing SPP algorithms with respect to large instances in each benchmark cluster within specified time limits (measured in hours). The CHDP algorithm is the constructive heuristic algorithm implemented in conjunction with the non-increasing perimeter sorting strategy of Leung et al. [116], while the IHRDH is the improved heuristic recursively implemented in conjunction with the non-increasing height sorting strategy of Zhang et al. [164]. The Hybrid GA is a combination of a GA with the CH algorithm implemented according to the recommendation provided in Chapter 11, and the IAm is the modified improved algorithm of Chapter 8.*

12.5 Chapter Summary

The relative effectiveness of the SPP metaheuristics considered in this dissertation was compared in this chapter in respect of the clustered benchmark instances of Chapter 7. The chapter opened in §12.1 with a description of the method of evaluation performed. Thereafter, the parameter settings selected for some of the metaheuristic implementations were presented in §12.2. A discussion followed in §12.3 on the comparative study results obtained by the various algorithms with respect to both solution quality and execution time. A characterisation of the

relative effectiveness of the various metaheuristics in respect of large SPP instances in the four benchmark clusters was finally provided in §12.4 under limited execution time frames.

The results of the comparative study of this chapter show that the newly proposed IAm algorithm is an improvement on the IA algorithm of Wei *et al.* [158], while the newly proposed SPSAL algorithm is an improvement on the SPGAL algorithm of Bortfeldt [25]. The IAm algorithm performs well on average and it achieves high-quality packing solutions within reasonable time frames. The results obtained in this study also indicate that the characteristics of the SPP benchmark instances employed may affect the mean solution quality achieved by the various packing algorithms. It is, therefore, advisable to take into consideration such attributes when conducting a comparative algorithmic study in order to avoid biased conclusions with respect to the algorithms' relative effectiveness.

Part V

Conclusion

CHAPTER 13

Dissertation Summary

Contents

This chapter contains a summary of the research documented in this dissertation, highlighting in §13.1 how the ten objectives presented in §1.4 were accomplished, and culminating in an appraisal in §13.2 of the novel contributions made in the dissertation.

13.1 Summary of Dissertation Contents

In the introductory chapter, a brief general background on C&P problems was provided, highlighting the relevance of these problems in real-world applications. This was followed by a description of the specific C&P problem considered in this dissertation, namely the 2D SPP. Thereafter, the research aim was described, after which the various underlying research objectives to be pursued were discussed. The chapter closed with an outline of the dissertation layout.

In pursuit of Objectives I–III of §1.4, a three-chapter part, Part I, was devoted to a review of the relevant literature on C&P problems as well as on existing algorithmic approaches toward solving instances of the specific type of C&P problem under consideration. The first chapter of this part, Chapter 2, contained a comprehensive overview of the literature on C&P problems, culminating in a delimitation of the scope of the dissertation. The chapter opened with a description of the various classifications of existing types of C&P problems in the literature, in fulfilment of Objectives I(a) and I(b). This included a presentation of the most prominent typologies for C&P problems in terms of the various characteristics of these problems. Thereafter, a thorough literature review on solution methodologies typically employed to solve C&P problems was provided, in pursuit of Objective II. The class of exact packing solution approaches was first discussed in this section, in fulfilment of Objective II(a). This was followed by a detailed description of the class of heuristic packing solution approaches (in fulfilment of Objective II(b)) and of the class of metaheuristic packing solution approaches (in pursuit of Objective II(c)). The type of C&P problem and the class of solution methodologies investigated in this dissertation were also defined.

In the following chapter, Chapter 3, the second chapter of the literature review part, five existing SPP heuristics were described in some detail, in pursuit of Objective III(a). After a brief

introduction, the modified best-fit decreasing height (BFDH*) algorithm of Bortfeldt [25] was described in prose form and by means of a pseudocode listing, after which a worked example was provided. This was followed by a description of the bottom-left algorithm of Liu and Teng [118]. The latter algorithm was described in the same manner as the BFDH* algorithm, and was further detailed by means of useful practical suggestions in terms of its algorithmic implementation. Thereafter, the improved heuristic recursive (IHR) algorithm proposed by Zhang *et al.* [164] was described and a pseudocode representation was provided together with an example of the working of the algorithm. The best-fit algorithm proposed by Burke *et al.* [29] was next presented in a similar manner. The last SPP heuristic described in the chapter was the constructive heuristic algorithm of Leung *et al.* [116].

The third chapter in the literature review part, Chapter 4, was dedicated to SPP metaheuristics, in fulfilment of Objective III(b) of §1.4. The reader was introduced to the basic working of two popular general metaheuristic search techniques, namely GAs and the method of SA, both described in the context of C&P problems. Various problem-specific and generic parameters and variables involved in each technique, which are typically employed in the realm of C&P problems, were reviewed. The next section of the chapter was devoted to detailed descriptions of a representative sample of well-known SPP metaheuristics. Strip packing solution approaches in the class of hybrid metaheuristics were first documented. These included a hybrid GA and a hybrid SA technique. Thereafter, the SPGAL algorithm of Bortfeldt [25] was described. The original CLP-GA algorithm of Bortfeldt and Gehring [26] for the container loading problem was considered in order to provide context, before a written description of the SPGAL algorithm was provided in some detail. This was followed by a description of the reactive GRASP algorithm of Alvarez-Valdés *et al.* [5]. Experimental studies carried out by the latter authors in respect of choosing the best elements for inclusion in the algorithm were covered in the description and a pseudocode listing of the procedure, together with a worked example, was also provided. A description of the two-stage intelligent search algorithm proposed by Leung *et al.* [116] followed that of the reactive GRASP algorithm, which was presented in a pseudocode form and illustrated by means of an example. Finally, the two improved ISA algorithms, namely the simple randomised algorithm of Yang *et al.* [162] and the improved algorithm of Wei *et al.* [158], were described in some detail.

Chapter 5, the first chapter of the second three-chapter part of the dissertation (devoted to the collection, documentation, and clustering of the SPP benchmark instances available in the literature), contained detailed descriptions of the SPP benchmark data instances employed throughout the dissertation for evaluating the relative performances of the various SPP algorithms under consideration, in fulfilment of Objective IV of §1.4. Two classes of benchmark instances were considered. Benchmark instances in the first class consist of zero-waste problem instances for which the respective optimal packing solutions are known and do not contain wasted areas (areas of the strip not occupied by items). This class of problem instances contains nine data sets, including the J instances of Jakobs [99], the SCP instances of Hifi [81], the babu instances of Babu and Babu [8], the NT and T instances of Hopper and Turton [91, 92], the N instances of Burke *et al.* [29], the CX instances of Pinto and Oliveira [133], and the IY instances of Imahori and Yagiura [94]. The second class of benchmark data instances consists of non-zero-waste instances for which optimal solutions are not known in all cases and those with optimal solutions known, but involving some wasted regions. This second class of benchmark instances contains eleven data sets, namely the cgut instances of Christofides and Whitlock [40], the beng instances of Bengtsson [18], the gcut and ngcut instances of Beasley [14, 15], the bwmv instances of Berkey and Wang [21] and of Martello and Vigo [123], the DP instances of Dagli *et al.* [48, 138], the BK instances of Burke and Kendall [28], the SCPL instances of Hifi [83], the nice and path instances of Valenzuela and Wang [156], the AH instances of Bortfeldt and Gehring [27], and

the zdf instances of Leung and Zhang [115]. The problem generators and methods employed to generate each of the aforementioned benchmark instances were also reviewed.

The second chapter of Part II, Chapter 6, was dedicated to a review of cluster analysis and contained relevant information on the type of clustering techniques and methods of clustering validation employed during the clustering study performed on the available SPP benchmark instances documented in Chapter 5, in fulfilment of Objective V of §1.4. The chapter opened with an overview of the topic of clustering in which a general background and descriptions of typical clustering processes were provided. This was followed by a presentation of the most prominent examples of clustering algorithms in the literature. These included the class of hierarchical clustering algorithms, the family of partitional clustering algorithms, the group of spectral clustering techniques, and the class of density-based clustering algorithms. Various clustering validation measures were also reviewed. In particular, four widely used clustering validation indices, namely the silhouette coefficient [139], the Caliński-Harabasz index [33], the Dunn index [54], and the Davies-Bouldin index [49], were described in some detail.

Details of the cluster analysis performed in respect of the SPP benchmark data of Chapter 5 were presented in the last chapter of Part II, Chapter 7, in pursuit of Objective V. The first section of the chapter contained a brief description of the data categorisation, highlighting the various features selected to describe the data. In the following section, the clustering process and the clustering result assessment were discussed. The second section opened with a presentation of the data preparation process, involving feature scaling of the benchmark data set. Thereafter, a visual inspection of the data was performed in order to ascertain whether or not the data exhibit natural clusters. This was achieved by presenting the data in the form of a scatter plot and performing PCA. It was found that the benchmark data did indeed exhibit natural clusters. This led to an estimation of the possible number of clusters that prevailed in the benchmark data. The **R** package, *NbClust*, was utilised for this purpose, and it was found that four was the best number of clusters. Finally, the process of selecting the most suitable clustering algorithm for classifying the data and the clustering output generated were discussed. In the last section, detailed descriptions were given of the characteristics of the various clusters of benchmark data.

During a preliminary computational study, conducted in fulfilment of Objective VI of §1.4, and involving implementation and evaluation of the various SPP metaheuristics reviewed in Chapter 4 in respect of the clustered benchmark data of Chapter 7, it became clear that some improvements could be made to certain algorithms. Two adaptations were suggested accordingly, and these algorithms were described in the first chapter of the third part of this dissertation, Chapter 8, devoted to new strip packing algorithms. This chapter was dedicated to descriptions of the working of these two adapted algorithms, in fulfilment of Objective VII(a) of §1.4. The first section of the chapter contained a detailed description of the first algorithmic improvement, the IAm algorithm. This algorithm was based on the IA algorithm of Wei *et al.* [158]. Several essential improvements were incorporated in the original algorithm in order to improve its performance. The respective details were provided in the aforementioned section, after which the working of the IAm algorithm was described in prose form and by means of a pseudocode representation. Another improved metaheuristic, the SPSAL algorithm, was described in the second section of the chapter. This algorithm was an adaptation of the SPGAL algorithm of Bortfeldt [25], involving application of the method of SA directly in the space of completely defined packing layouts (*i.e.* without any encoding of solutions). A detailed description was provided of the working of the algorithm, together with a pseudocode representation of the overall procedure.

Since the two newly proposed algorithmic adaptations of Chapter 8 were both SA packing techniques, their implementations required suitable values for the integrated SA parameters. A

computational study, based on an experimental design, was conducted for this purpose, aimed at identifying the best combination of the algorithmic parameter values in each case. Details of the evaluation study performed, as well as the results obtained, were presented in the second chapter of Part III, Chapter 9, in fulfilment of Objectives VII(b) and VIII of §1.4. The chapter opened with a description of the performance evaluation measures employed and the type of statistical analysis performed. This was followed by a presentation of the specific implementation of the method of SA employed in the two algorithms, and included discussions on the method utilised to calculate appropriate values for the initial temperature parameter, the type of cooling schedule implemented, and the epoch management rules adopted. Thereafter, details were presented of the experimental design followed. Finally, the computational results obtained in respect of the clustered SPP benchmark instances of Chapter 7 were reported in the form of boxplots substantiated with appropriate statistical analyses.

The following chapter, Chapter 10, was devoted to an appraisal of the five SPP heuristics reviewed in Chapter 3 in terms of the solution qualities they yield, the effect of different sorting strategies on the packing solutions they yield, and their execution times, in pursuit of Objective IX(a) of §1.4. This chapter is the first chapter of Part IV of the dissertation, devoted to an evaluation of the relative effectivenesses of the various SPP algorithmic approaches under investigation. Each of the five heuristic algorithms was implemented employing four different sorting strategies, and the packing solution qualities achieved were compared in respect of the clustered benchmark instances of Chapter 7. The results obtained were presented and interpreted in the first section of the chapter. The next section contained the comparative study results of the five heuristics in respect of the benchmark clusters. This comparison took place in respect of the best implementation of each heuristic, as determined during the computational study of the previous section, and was carried out at a 95% level of confidence in terms of solution quality. The algorithmic execution times were also reported and compared. It was concluded that the packing order has a crucial effect on the performance of the different heuristics. Moreover, the characteristics of the benchmark instances affect the mean solution quality achieved by the five packing heuristics.

Chapter 11, the second chapter of Part IV, was dedicated to descriptions of a limited computational study with respect to implementations of the known hybrid metaheuristics described in Chapter 4, in fulfilment of Objective IX(b) of §1.4. The aim of the chapter was to identify superior implementations of the GA and the method of SA employed in the hybrid GA and hybrid SA SPP algorithms in terms of their constituent components (operators and parameter values). The computational study consisted of an experimental design and a sensitivity analysis in which various parameter combinations of each metaheuristic implementation were evaluated and tested, as described in the first section of the chapter. The experimental design followed to determine the best combination parameter values for the GA was presented in the following section. The experimental design involved two separate phases. During the first phase, different types of crossover operators, selection procedures, and replacement techniques were combined and evaluated. During the second phase, the best combinations of parameter values obtained during the first phase were kept constant, while varying the crossover rate parameter and the fitness function. Detailed descriptions of the experiments, as well as the results obtained in respect of the clustered benchmark instances of Chapter 7, were presented in the above-mentioned section. A similar experimental design was also performed in respect of the method of SA for the hybrid SA algorithm and detailed descriptions of the experiments, together with the results obtained, were discussed in the third section of the chapter. The first experimental design phase in this case involved evaluation of three parameters, namely the initial temperature parameter, the cooling parameter, and the epoch management parameter, while the second phase involved variation of the fitness function parameter. The most effective algorithmic implementations

based on the best parameter combination values for both the hybrid GA and hybrid SA algorithms were described in the last section of the chapter with respect to each cluster of benchmark instances separately. The results obtained in this chapter suggested that:

- A hybrid GA implementation employing elitism replacement is statistically suggested for implementation in respect of all four benchmark clusters. The hybrid GA is not sensitive with respect to the crossover operator and selection technique employed.
- The use of packing height as fitness function is preferable in the algorithmic implementation of the hybrid GA, and a value 0.6 of the crossover rate is suggested for the hybrid GA implementation in respect of all four benchmark clusters.
- A relatively small value of the initial temperature (namely 29.79) is statistically recommended for adoption in the hybrid SA algorithm in respect of all four benchmark clusters.
- A cooling parameter value of 0.93 is preferred for the SPP instances in Clusters 1 and 4, whereas a value of 0.95 is better suited for implementation in the hybrid SA for the instances in Clusters 2 and 3.
- A fixed number of N iterations, where N denotes the problem dimension, is suggested as epoch termination trigger for the instances in Clusters 1–3, while an epoch should be terminated when $\frac{N}{2}$ successful moves have been attempted during the search in the case of benchmark instances in Cluster 4.
- The utilisation of packing height as fitness function is also preferable in the algorithmic implementation of the hybrid SA in respect of all four benchmark clusters.

The relative effectivenesses of all the SPP metaheuristics considered, namely the seven algorithms reviewed in Chapter 4 and the two newly proposed algorithmic adaptations of Chapter 8, were finally compared in Chapter 12, the last chapter of Part IV, in respect of the clustered benchmark instances of Chapter 7 in fulfilment of Objective IX(c). The chapter opened with a presentation of the method of comparison and evaluation performed. Thereafter, detailed descriptions were provided of how the various algorithms were implemented. These included discussions on the parameter settings selected for the SPGAL, the reactive GRASP, the ISA, the SRA, and the IA implementations, and on the simulations performed in respect of each algorithm. This was followed by a presentation of the comparative study results of the nine algorithms, again carried out at a 95% level of confidence in terms of solution quality. The various algorithmic execution times were also reported and compared. The following conclusions were reached based on this appraisal:

- The newly proposed IAm algorithm performs well on average, competing favourably with the known algorithms in terms of both solution quality and execution time.
- The IAm algorithm is an improvement on the IA algorithm of Wei *et al.* [158], while the newly proposed SPSAL algorithm is an improvement on the SPGAL algorithm of Bortfeldt [25].
- The mean performance ratio ranks of the various algorithms change, depending on the SPP benchmark clusters to which they are applied. That is, some algorithms yield better results than others with respect to certain benchmark clusters, whereas the same algorithms achieve worse results for other benchmark instances.
- The best hybrid GA implementation, identified in Chapter 11, performs better than the other metaheuristics in respect of some SPP instances. Furthermore, a hybrid GA algorithm is superior to a hybrid SA algorithm in respect of all four benchmark clusters.

- The hybrid SA algorithm is the fastest algorithm, followed by the IAm and SPGAL algorithms, while the IA algorithm is the slowest algorithm.

A characterisation of the time efficiencies of the various SPP algorithms in respect of large SPP instances in each of the benchmark clusters was provided in the last section of Chapter 12. The CH algorithm of Leung *et al.* [116] is statistically suggested as the SPP algorithm of choice in respect of the Cluster 1, 3 and 4 benchmark instances if the execution time budget is less than 1 hour, while the IHR algorithm of Zhang *et al.* [164] is preferred for the instances in Cluster 2 within a similar time frame. If the time available for solving an SPP instance is around 2 hours, the IAm algorithm should be selected as the packing solution approach for all four benchmark clusters. If, however, the time budget is more than 5 hours, the hybrid GA algorithm is recommended for instances in Clusters 2 and 3, while the IAm algorithm should rather be selected for benchmark instances in Clusters 1 and 4.

Finally, the results of this study demonstrated that the underlying characteristics of the benchmark instances may affect the mean solution qualities and the relative rankings of the various SPP algorithms. It is, therefore, recommended that the characteristics of the benchmark instances considered for algorithmic evaluation purposes should be taken into account in order to avoid biased conclusions with respect to the relative effectiveness of the algorithms.

13.2 Appraisal of Dissertation Contributions

This section contains an appraisal of the contributions made in this dissertation. The contributions are six-fold and they are listed here in the order in which they appear in the dissertation. In each case the contribution is discussed briefly in terms of its value.

Contribution 1 *A review of existing heuristics and the state-of-the-art metaheuristics in the literature for solving the 2D SPP.*

The overarching aim in this dissertation was to propose new 2D SPP metaheuristics that improve on the performance of existing methods in the literature in terms of both solution quality and execution time. In order to achieve this, a review of the literature on 2D SPP heuristics and metaheuristics was performed. A general overview of the field of C&P problems was provided in Chapter 2 with a view to place the topic of the dissertation in context. Five well-known 2D SPP heuristics from the literature were also described in detail in Chapter 3 and seven state-of-the-art 2D SPP metaheuristics were presented in Chapter 4. This review was fundamental in developing an understanding of the main concepts considered during the past few decades for solving instances of the 2D SPP, and was also important in terms of providing practical guidelines for the various algorithmic implementations.

Contribution 2 *The clustering, for the first time, of the available SPP benchmark instances in the literature into different classes of test problems.*

Significant research gaps were identified in the literature, mainly associated to the use of relatively small benchmark instances in respect of comparing the performances of packing methods and a lack of acknowledgment that the characteristics of these instances may affect the mean packing solution qualities achieved by the various methods. An attempt was thus made in this dissertation to collect all available SPP benchmark instances in the literature and to categorise

these data into different groups of test problems based on their underlying features. A total of 1 718 benchmark problem instances were collected for this purpose, all described in Chapter 5, and a clustering analysis was performed in Chapter 7 in respect of these data. Four natural benchmark clusters were obtained accordingly. It is envisaged that these clustered benchmark instances may be adopted in the future in the context of SPP methodological performance comparisons.

Contribution 3 *Two new metaheuristic algorithms, based on the method of SA, for solving instances of the 2D SPP.*

During a pilot computational study involving the implementation and evaluation of the various SPP metaheuristics reviewed in Chapter 4, a number of shortcomings became apparent and some improvements could therefore be made to certain algorithms. Two improved SPP metaheuristics (referred to as the IAm and the SPSAL algorithms) were therefore proposed in Chapter 8 in order to address these shortcomings. The IAm algorithm was based on the IA algorithm of Wei *et al.* [158], involving a combination of the method of SA with a heuristic construction algorithm, while the SPSAL algorithm was an adaptation of the SPGAL algorithm of Bortfeldt [25], involving an application of the method of SA to solve SPP instances directly in the space of the completely defined packing layouts.

Contribution 4 *A statistical comparison of the relative performances of five strip packing heuristics in terms of both solution quality and execution time in respect of the clustered benchmark instances of Contribution 2.*

To the best knowledge of the author, the effects of the characteristics of the benchmark instances on the performances of various SPP algorithms have not been considered prior to this study. In Chapter 10 of this dissertation, this limitation was addressed: The effectiveness of the designs of the five SPP heuristics reviewed in Chapter 3 were compared and contrasted for each of the data clusters of Chapter 7. The comparisons were based on statistical analyses and conclusions were drawn from these analyses at a 95% level of confidence. The results obtained in this study demonstrated that the characteristics of the benchmark instances indeed affect the mean solution qualities achieved by the various algorithms, which led to the recommendation that these characteristics should be taken into account during comparative algorithmic studies in the future in order to avoid biased research conclusions.

Contribution 5 *A parameter settings analysis of known hybrid GA and hybrid SA SPP techniques in respect of the clustered benchmark instances of Contribution 2.*

It has been reported in the literature that the choice of certain parameter values in the metaheuristic techniques employed in hybrid SPP algorithms influences the performances of these algorithms [91, 116]. The computational study, based on an experimental design and a sensitivity analysis in respect of the various parameter settings of hybrid GA and hybrid SA SPP techniques, conducted in Chapter 11, is therefore considered an important contribution of this dissertation. Extensive parameter optimisation experiments were performed during this study in order to find the best combination of GA and SA parameters for the four benchmark clusters of Chapter 7. The most effective algorithmic implementations for both the hybrid GA and hybrid SA algorithms with respect to each cluster of benchmark instances were thus provided.

Contribution 6 *A statistical comparison of the relative performances of the two improved strip packing metaheuristics of Contribution 3 with seven existing metaheuristics in terms of both solution quality and execution time in respect of the clustered benchmark instances of Contribution 2.*

The significant value of Contribution 3 was emphasised in Chapter 12 of this dissertation by conducting a computational study involving a comparison of the relative effectiveness of the two newly proposed algorithms and the seven well-known metaheuristics of Chapter 4. The comparison was carried out at a 95% level of confidence in terms of solution quality in respect of the clustered benchmark instances of Chapter 7. The various algorithmic execution times were also compared. The results thus obtained revealed that the newly proposed IAm algorithm performs well on average and that it achieves high-quality packing solutions within reasonable time frames. Furthermore, the newly proposed SPSAL algorithm is an improvement on the SPGAL algorithm of Bortfeldt [25]. A similar recommendation as in Contribution 4 was also drawn from the results: The underlying characteristics of the benchmark instances employed for comparative algorithmic study purposes should certainly be taken into account in the future.

CHAPTER 14

Future Work

Contents

This chapter contains six suggestions for possible future pursuit as follow-up work on the research carried out in this dissertation. A brief description is provided in each case.

14.1 Additional Benchmark Data Analyses

This section contains two suggestions for future work related to the SPP benchmark data analysis performed in this dissertation.

Suggestion 1 *Identify additional benchmark data features.*

Four features were considered in this study to categorise the SPP benchmark data into different classes of test problems. These features were selected based on the parameters and characteristics produced by the most popular problem generators. It is suggested that further features be added to complement the characterisation of the problem benchmark instances. Possible additional features may include the *maximum perimeter ratio*, which is the maximum of the perimeter ratio of any pair of items in a given SPP instance, or the *size proportional ratio*, which is the ratio between the number of wide items (with width larger than half of the strip width) and the number of narrow items (with width at most half of the strip width) in an SPP instance. The inclusion of other features is expected to result in a more robust classification of the various problem instances.

In the same vein, methodologies capable of extracting features from the benchmark data automatically may also be considered. It might be beneficial to employ machine learning techniques, including deep learning, to identify and select the most significant characteristics of the benchmark data. A methodology based on linear correlations and PCA has been employed by Júnior *et al.* [103] to identify characteristics of 2D-SPP benchmark instances, but it would be interesting to employ alternative techniques that are capable of extracting relevant and important problem characteristics, avoiding redundancy.

Suggestion 2 *Test additional SPP problem instances.*

Possible future work might also involve incorporating additional test problems in the analyses carried out in this dissertation. The author is aware of other existing problem generators, such as the 2DCPackGen of Silva *et al.* [143], which may be employed to generate various problem instances. Incorporating such additional test problems in the clustering analysis of Chapter 7 may facilitate identification of other important features prevailing in the data, and also render the methodology more realistic and robust in the sense of being able to accommodate a large variety of problem instances.

It is also suggested that a user-friendly computerised decision support system, capable of solving SPP instances and based on the clustered benchmark instances, is designed to make the various algorithmic implementations considered in this dissertation accessible to users or industry practitioners. Such software should take as input a new problem instance. After reading in the given problem instance, it should apply clustering and assign the instance to a relevant cluster. Thereafter, it should perform the respective packing task by means of an appropriate algorithm, as recommended in Table 12.7, and report the results as output to the user.

14.2 Alternative SPP Solution Techniques

This section contains two further suggestions for future work related to the SPP solution methodologies considered in this dissertation.

Suggestion 3 *Improve upon the SPSAL algorithm.*

While the SPSAL of §8.2 improves upon the SPGAL algorithm proposed by Bortfeldt [25], there is still further room for improvement in respect of this algorithm. The heuristic employed to generate a starting solution and the post-optimisation heuristic performed at the end of the procedure are the key aspects of this algorithm in terms of generating relatively superior packing solutions. A greedy selection procedure, employing the BFDH* heuristic of Bortfeldt [25], was employed in the SPSAL algorithm to generate an initial solution. It is suggested that the BFDH* algorithm be replaced by a better performing level-based heuristic, such as the IHR algorithm of Zhang *et al.* [164], so as to improve upon the quality of the packing solution generated at the beginning of the search. It is also suggested that the post-optimisation procedure be performed at the end of each iteration during the execution of the algorithm. That is, after performing the CLP-SA of Algorithm 8.2, the post-optimisation process should be applied directly in order to reduce the packing height. The new packing height thus obtained may then be used during the next iteration of the procedure instead of attempting to reduce the previous container length by one unit. This might be beneficial in terms of achieving a high-quality packing solution within a reasonable time.

Suggestion 4 *Design a hyper heuristic and/or hyper algorithm.*

It was found in §12.3.1 that the IAm algorithm performed better than the hybrid GA in respect of the benchmark instances of Clusters 1 and 4, while the opposite result was found in respect of the other two benchmark clusters. It was similarly found in §10.2 that the IHR algorithm outperformed the other four heuristics in respect of Cluster 2 data, while the CH algorithm yielded the best results for the remaining benchmark clusters. It is therefore suggested that

combinations of algorithms be used to solve instances of the 2D SPP in an attempt at exploiting the most favourable characteristics of each algorithm. This may be achieved by employing a hyper-algorithm which decides which of a number of algorithms, such as the IAm algorithm and the hybrid GA, should be selected is respect of a given SPP instance.

14.3 Application to other Types of C&P Problems

This section contains two final suggestions for future work related to natural generalisations of the techniques assessed in this dissertation.

Suggestion 5 *Adapt the SPSAL and IAm algorithms to accommodate rotation of items.*

The two newly proposed algorithms of Chapter 8 may be adapted for instances of the SPP in which rotation of items is allowed. A number of options are suggested here as to how the facilitation of item rotation may be incorporated in these two algorithms. In the IAm algorithm, rotation may be accommodated in the form of a manipulation operator during the generation of a new neighbouring solution. That is, the neighbourhood solutions of the current solution may be reached by applying either a swapping rule (interchanging the order of two randomly selected items in the packing permutation) or an orientation rule (rotating one randomly selected item). Rotation may also be achieved during the selection of the most suitable item for packing in the IAm algorithm. That is, during the scoring process of an item, both of its orientations may be evaluated and assigned scores. The orientation yielding the largest score may then be considered for selection.

In the case of the SPSAL algorithm, rotation of items may be accommodated during the post-optimisation procedure. As described in §4.5, the post-optimisation heuristic consists of arranging a full packing layout into consecutive block layers, then reorganising each block layer so that empty spaces are identified, and finally displacing certain layers to fill in gaps. It is suggested that the rotation of items may be embedded in the reorganisation phase or at the end of the displacement phase in order to improve upon the performance of the algorithm.

Suggestion 6 *Apply the methodology to other types of C&P problems.*

A noteworthy finding in this dissertation was the necessity of considering the characteristics of the benchmark data employed during a comparative algorithmic study. It was recommended that these characteristics should be taken into account in the future to avoid biased research conclusions. For a similar reason, it is suggested that such aspects be considered in other types of C&P problems, such as the bin packing problem and cutting stock problem. The same analysis framework and clustering analysis as considered in this dissertation may be applied to these problems.

References

[1] Aggarwal CC & Reddy CK, 2013, *Data clustering: Algorithms and applications*, CRC Press, New York (NY).

[2] Agico Group, 2016, *Steel cutting tools*, [Online], [Cited December 2017], Available from `http://www.sellsteels.com/service/cutting-steel.html`.

[3] Alvarez-Valdés R, Parajón A & Tamarit JM, 2002, *A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems*, Computers and Operations Research, **29(7)**, pp. 925–947.

[4] Alvarez-Valdés R, Parreño F & Tamarit JM, 2005, *A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems*, Journal of the Operational Research Society, **56(4)**, pp. 414–425.

[5] Alvarez-Valdés R, Parreño F & Tamarit JM, 2006, *Reactive GRASP for the strip-packing problem*, Computers and Operations Research, **35(4)**, pp. 1065–1083.

[6] Alvarez-Valdés R, Parreño F & Tamarit JM, 2007, *A tabu search algorithm for a two-dimensional non-guillotine cutting problem*, European Journal of Operational Research, **183(3)**, pp. 1167–1182.

[7] Anderson GB, 2013, *Principal component analysis in R*, [Online], [Cited July 2017], Available from `https://www.ime.usp.br/~pavan/pdf/MAE0330-PCA-R-2013`.

[8] Babu AR & Babu NR, 1999, *Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms*, International Journal of Production Research, **37(7)**, pp. 1625–1643.

[9] Bach FR & Jordan MI, 2004, *Learning spectral clustering*, Proceedings of the Advances in Neural Information Processing Systems, Vancouver, pp. 305–312.

[10] Baker BS, Brown DJ & Katseff HP, 1981, *A 54 algorithm for two-dimensional packing*, Journal of Algorithms, **2(4)**, pp. 348–368.

[11] Baker BS, Coffman Jr EG & Rivest RL, 1980, *Orthogonal packings in two dimensions*, SIAM Journal on Computing, **9(4)**, pp. 846–855.

[12] Baker JE, 1987, *Reducing bias and inefficiency in the selection algorithm*, Proceedings of the 2nd International Conference on Genetic Algorithms, Hillsdale (NJ), pp. 14–21.

[13] Bansal N, Oosterwijk T, Vredeveld T & Van Der Zwaan R, 2016, *Approximating vector scheduling: Almost matching upper and lower bounds*, Algorithmica, **76(4)**, pp. 1077–1096.

[14] Beasley J, 1985, *Algorithms for unconstrained two-dimensional guillotine cutting*, Journal of the Operational Research Society, **36(4)**, pp. 297–306.

[15] Beasley J, 1985, *An exact two-dimensional non-guillotine cutting tree search procedure*, Operations Research, **33(1)**, pp. 49–64.

[16] Bekrar A & Kacem I, 2009, *An exact method for the 2D guillotine strip packing problem*, [Online], [Cited July 2017], Available from `https://www.hindawi.com/journals/aor/2009/732010/`.

[17] Belov G & Scheithauer G, 2006, *A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting*, European Journal of Operational Research, **171(1)**, pp. 85–106.

[18] Bengtsson B, 1982, *Packing rectangular pieces — A heuristic approach*, The Computer Journal, **25(3)**, pp. 353–357.

[19] Bennell JA & Oliveira JF, 2008, *The geometry of nesting problems: A tutorial*, European Journal of Operational Research, **184(2)**, pp. 397–415.

[20] Bennell JA & Oliveira JF, 2009, *A tutorial in irregular shape packing problems*, Journal of the Operational Research Society, **60(1)**, pp. S93–S105.

[21] Berkey JO & Wang PY, 1987, *Two-dimensional finite bin-packing algorithms*, Journal of the Operational Research Society, **38(5)**, pp. 423–429.

[22] Berkhin P, 2006, *A survey of clustering data mining techniques*, pp. 25–71 in Kogan J, Nicholas C & Teboulle M (Eds), *Grouping multidimensional data*, Springer, Berlin.

[23] Blickle T & Thiele L, 1995, *A comparison of selection schemes used in genetic algorithms*, Computer Engineering and Networks Laboratory (TIK) Report, Zurich.

[24] Bologna ORG, 2017, *Library of instances*, [Online], [Cited July 2017], Available from `http://or.dei.unibo.it/library/two-dimensional-bin-packing-problem`.

[25] Bortfeldt A, 2006, *A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces*, European Journal of Operational Research, **172(3)**, pp. 814–837.

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

[27] Bortfeldt A & Gehring H, 2006, *New large benchmark instances for the two-dimensional strip packing problem with rectangular pieces*, Proceedings of the 39th Annual Hawaii International Conference on System Sciences, Kauai (HI), p. 30.

[28] Burke EK & Kendall G, 1999, *Applying simulated annealing and the no fit polygon to the nesting problem*, Proceedings of the World Manufacturing Congress, Durham, pp. 27–30.

[29] Burke EK, Kendall G & Whitwell G, 2004, *A new placement heuristic for the orthogonal stock-cutting problem*, Operations Research, **52(4)**, pp. 655–671.

[30] Burke EK, Kendall G & Whitwell G, 2006, *Metaheuristic enhancements of the best-fit heuristic for the orthogonal stock cutting problem*, (Unpublished) Technical Report NOTTCS-TR-SUB-0605091028-4370, Computer Science Department, University of Nottingham, Nottingham.

[31] Burke EK, Kendall G & Whitwell G, 2009, *A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock-cutting problem*, INFORMS Journal on Computing, **21(3)**, pp. 505–516.

[32] Busetti F, 2003, *Simulated annealing overview*, [Online], [Cited August 2017], Available from `http://www.aiinfinance.com/saweb.pdf`.

[33] Caliński T & Harabasz J, 1974, *A dendrite method for cluster analysis*, Communications in Statistics-theory and Methods, **3(1)**, pp. 1–27.

[34] Cattell RB, 1943, *The description of personality: Basic traits resolved into clusters*, Journal of Abnormal and Social Psychology, **38(4)**, pp. 476–506.

[35] Chambers LD, 2000, *The practical handbook of genetic algorithms: Applications*, Chapman and Hall/CRC, New York (NY).

[36] Charrad M, Ghazzali N, Boiteau V, Niknafs A & Charrad MM, 2014, *Package NbClust*, Journal of Statistical Software, **61**, pp. 1–36.

[37] Chazelle B, 1983, *The bottomn-left bin-packing heuristic: An efficient implementation*, IEEE Transactions on Computers, **100(8)**, pp. 697–707.

[38] Chekuri C & Khanna S, 2004, *On multi-dimensional packing problems*, SIAM Journal on Computing, **33(4)**, pp. 837–851.

[39] Chen Y & Tu L, 2007, *Density-based clustering for real-time stream data*, Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose (CA), pp. 133–142.

[40] Christofides N & Whitlock C, 1977, *An algorithm for two-dimensional cutting problems*, Operations Research, **25(1)**, pp. 30–44.

[41] Chung FR, Garey MR & Johnson DS, 1982, *On packing two-dimensional bins*, SIAM Journal on Algebraic Discrete Methods, **3(1)**, pp. 66–76.

[42] Coffman Jr EG, Garey MR, Johnson DS & Tarjan RE, 1980, *Performance bounds for level-oriented two-dimensional packing algorithms*, SIAM Journal on Computing, **9(4)**, pp. 808–826.

[43] Coffman Jr EG, Johnson DS, Lueker G & Shor P, 1993, *Probabilistic analysis of packing and related partitioning problems*, Statistical Science, **28**, pp. 40–47.

[44] Coffman Jr EG & Shor P, 1990, *Average-case analysis of cutting and packing in two dimensions*, European Journal of Operational Research, **44(2)**, pp. 134–144.

[45] Conway JH & Sloane NJA, 2013, *Sphere packings, lattices and groups*, Springer Science and Business Media, New York (NY).

[46] Cormack RM, 1971, *A review of classification*, Journal of the Royal Statistical Society: Series A (General), **134(3)**, pp. 321–367.

[47] Cui Y, Yang Y, Cheng X & Song P, 2008, *A recursive branch-and-bound algorithm for the rectangular guillotine strip packing problem*, Computers and Operations Research, **35(4)**, pp. 1281–1291.

[48] Dagli CH & Poshyanonda P, 1997, *New approaches to nesting rectangular patterns*, Journal of Intelligent Manufacturing, **8(3)**, pp. 177–190.

[49] Davies DL & Bouldin DW, 1979, *A cluster separation measure*, IEEE Transactions on Pattern Analysis and Machine Intelligence, **(2)**, pp. 224–227.

[50] De Jong KA, 1975, *Analysis of the behavior of a class of genetic adaptive systems*, (Unpublished) Technical Report UMR0635, Computer and Communication Sciences Department, University of Michigan, Ann Arbor (MI).

[51] Demšar J, 2006, *Statistical comparisons of classifiers over multiple data sets*, Journal of Machine learning research, **7**, pp. 1–30.

[52] Dowsland KA, 1993, *Some experiments with simulated annealing techniques for packing problems*, European Journal of Operational Research, **68(3)**, pp. 389–399.

[53] Dowsland WB, 1984, *Two and three dimensional packing problems and solution methods*, [Online], [Cited June 2016], Available from `http://www.thebookshelf.auckland.ac.nz/docs/NZOperationalResearch/1985vol13/Number1/ORSNZ-1985-13-1-01.pdf`.

[54] Dunn JC, 1974, *Well-separated clusters and optimal fuzzy partitions*, Journal of Cybernetics, **4(1)**, pp. 95–104.

[55] Dyckhoff H, 1990, *A typology of cutting and packing problems*, European Journal of Operational Research, **44(2)**, pp. 145–159.

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

[57] Eglese R, 1990, *Simulated annealing: A tool for operational research*, European Journal of Operational Research, **46(3)**, pp. 271–281.

[58] Eisemann K, 1957, *The trim problem*, Management Science, **3(3)**, pp. 279–284.

[59] Elbatta MT & Ashour WM, 2013, *A dynamic method for discovering density varied clusters*, International Journal of Signal Processing, Image Processing, and Pattern Recognition, **6(1)**, pp. 123–134.

[60] ESICUP, 2015, *Datasets 2D-rectangular*, [Online], [Cited July 2017], Available from `http://paginas.fe.up.pt/~esicup/datasets?category_id=3`.

[61] Ester M, Kriegel HP, Sander J & Xu X, 1996, *A density-based algorithm for discovering clusters in large spatial databases with noise*, Proceedings of the 2^{nd} International Conference on Knowledge Discovery and Data Mining, Portland (OR), pp. 226–231.

[62] Faina L, 1999, *An application of simulated annealing to the cutting stock problem*, European Journal of Operational Research, **114(3)**, pp. 542–556.

[63] Feo TA & Resende MG, 1995, *Greedy randomized adaptive search procedures*, Journal of Global Optimization, **6(2)**, pp. 109–133.

[64] Gilmore PC & Gomory RE, 1961, *A linear programming approach to the cutting stock problem — Part I*, Operations Research, **9(6)**, pp. 849–859.

[65] Gilmore PC & Gomory RE, 1963, *A linear programming approach to the cutting stock problem — Part II*, Operations Research, **11(6)**, pp. 863–888.

[66] Gilmore PC & Gomory RE, 1965, *Multistage cutting stock problems of two and more dimensions*, Operations Research, **13(1)**, pp. 94–120.

[67] Glover F, 1986, *Future paths for integer programming and links to artificial intelligence*, Computers and Operations Research, **13(5)**, pp. 533–549.

[68] Glover FW & Kochenberger GA, 2006, *Handbook of metaheuristics*, Springer Science and Business Media.

[69] Golan I, 1981, *Performance bounds for orthogonal oriented two-dimensional packing algorithms*, SIAM Journal on Computing, **10(3)**, pp. 571–582.

[70] Goldberg DE & Lingle R, 1985, *Alleles, loci, and the traveling salesman problem*, Proceedings of the International Conference on Genetic Algorithms and their Applications, Hillsdale (NJ), pp. 154–159.

[71] Gomes AM & Oliveira JF, 2006, *Solving irregular strip packing problems by hybridising simulated annealing and linear programming*, European Journal of Operational Research, **171(3)**, pp. 811–829.

[72] Goyal SK, 1973, *Determination of economic packaging frequency for items jointly replenished*, Management Science, **20(2)**, pp. 232–235.

[73] Halkidi M, Batistakis Y & Vazirgiannis M, 2000, *Quality scheme assessment in the clustering process*, Proceedings of the Principles of Data Mining and Knowledge Discovery, Berlin, pp. 265–276.

[74] Halkidi M, Batistakis Y & Vazirgiannis M, 2001, *On clustering validation techniques*, Journal of Intelligent Information Systems, **17(2)**, pp. 107–145.

[75] Halkidi M, Batistakis Y & Vazirgiannis M, 2002, *Cluster validity methods: Part I*, ACM Sigmod Record, **31(2)**, pp. 40–45.

[76] Halkidi M, Batistakis Y & Vazirgiannis M, 2002, *Cluster validity methods: Part II*, ACM Sigmod Record, **31(3)**, pp. 19–27.

[77] Halkidi M & Vazirgiannis M, 2001, *Clustering validity assessment: Finding the optimal partitioning of a data set*, Proceedings of the International Conference on Data Mining, San Jose (CA), pp. 187–194.

[78] Hansen P & Jaumard B, 1997, *Cluster analysis and mathematical programming*, Mathematical Programming, **79(1-3)**, pp. 191–215.

[79] Hartigan JA & Hartigan J, 1975, *Clustering algorithms*, Wiley, New York (NY).

[80] He Q, 1999, *A review of clustering algorithms as applied in IR*, (Unpublished) Technical Report UIUCLIS–1999/6+IRG, Graduate School of Library and Information Science, University of Illinois, Urbana-Champaign (IL).

[81] Hifi M, 1998, *Exact algorithms for the guillotine strip cutting and packing problem*, Computers and Operations Research, **25(11)**, pp. 925–940.

[82] Hifi M, 2004, *Library of instances*, [Online], [Cited July 2017], Available from `ftp://cermsem.univ-paris1.fr/pub/CERMSEM/hifi/Strip-cutting/`.

[83] Hifi M, 1999, *The strip cutting and packing problem: Incremental substrip algorithms-based heuristics*, Pesquisa Operacional, **19(2)**, pp. 169–188.

[84] Hifi M & Zissimopoulos V, 1997, *Constrained two-dimensional cutting: An improvement of Christofides and Whitlock's exact algorithm*, Journal of the Operational Research Society, **48(3)**, pp. 324–331.

[85] Hinxman A, 1980, *The trim-loss and assortment problems: A survey*, European Journal of Operational Research, **5(1)**, pp. 8–18.

[86] Hochberg Y & Tamhane A, 1987, *Multiple comparison procedures*, John Wiley and Sons, New York (NY).

[87] Holland JH, 1975, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*, University of Michigan Press, Ann Arbor (MI).

[88] Hopper E, 2000, *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods*, PhD Dissertation, University of Wales, Cardiff.

[89] Hopper E & Turton BC, 1998, *Application of genetic algorithms to packing problems — A review*, pp. 279–288 in Chawdhry PK, Roy R & Pant RK (Eds), *Soft Computing in Engineering Design and Manufacturing*, Springer, London.

[90] Hopper E & Turton BC, 2001, *A review of the application of meta-heuristic algorithms to 2D strip packing problems*, Artificial Intelligence Review, **16(4)**, pp. 257–300.

[91] Hopper E & Turton BC, 2001, *An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem*, European Journal of Operational Research, **128(1)**, pp. 34–57.

[92] Hopper E & Turton BC, 2002, *Problem generators for rectangular packing problems*, Studia Informatica Universalis, **2(1)**, pp. 123–136.

[93] Hwang SM, Kao CY & Horng JT, 1994, *On solving rectangle bin packing problems using genetic algorithms*, Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, San Antonio (TX), pp. 1583–1590.

[94] Imahori S & Yagiura M, 2010, *The best-fit heuristic for the rectangular strip packing problem: An efficient implementation and the worst-case approximation ratio*, Computers and Operations Research, **37(2)**, pp. 325–333.

[95] Imahori S & Yagiura M, 2010, *Cutting and packing, test instances "Strip packing problem"*, [Online], [Cited July 2017], Available from `http://www-or.amp.i.kyoto-u.ac.jp/~imahori/packing/instance.html`.

[96] Jain AK, 2010, *Data clustering: 50 years beyond K-means*, Pattern Recognition Letters, **31(8)**, pp. 651–666.

[97] Jain AK & Dubes RC, 1988, *Algorithms for clustering data*, Prentice-Hall, Upper Saddle River (NJ).

[98] Jain AK, Murty MN & Flynn PJ, 1999, *Data clustering: A review*, ACM Computing Surveys (CSUR), **31(3)**, pp. 264–323.

[99] Jakobs S, 1996, *On genetic algorithms for the packing of polygons*, European Journal of Operational Research, **88(1)**, pp. 165–181.

[100] Johnson DS, 1974, *Fast algorithms for bin packing*, Journal of Computer and System Sciences, **8(3)**, pp. 272–314.

[101] Johnson DS, Demers A, Ullman JD, Garey MR & Graham RL, 1974, *Worst-case performance bounds for simple one-dimensional packing algorithms*, SIAM Journal on Computing, **3(4)**, pp. 299–325.

[102] Jolliffe IT, 1986, *Principal component analysis and factor analysis*, pp. 115–128 in Jolliffe IT (Ed), *Principal component analysis*, Springer, New York (NY).

[103] Júnior AN, Silva E, Gomes AM & Oliveira JF, 2017, *The two-dimensional strip packing problem: What matters?*, Proceedings of the 18th Congress of the Portuguese Operational Research Society, Cham, pp. 151–164.

[104] Kantorovich LV, 1960, *Mathematical methods of organizing and planning production*, Management Science, **6(4)**, pp. 366–422.

[105] Karami A & Johansson R, 2014, *Choosing dbscan parameters automatically using differential evolution*, International Journal of Computer Applications, **91(7)**, pp. 1–11.

[106] Kaufman L & Rousseeuw PJ, 2009, *Finding groups in data: An introduction to cluster analysis*, John Wiley and Sons, New York (NY).

[107] Kenmochi M, Imamichi T, Nonobe K, Yagiura M & Nagamochi H, 2009, *Exact algorithms for the two-dimensional strip packing problem with and without rotations*, European Journal of Operational Research, **198(1)**, pp. 73–83.

[108] Kidd M, 2018, Statistician in the Centre for Statistic Consultation at *Stellenbosch University*, [Personal Communication], Contactable at `mkidd@sun.ac.za`.

[109] Kirkpatrick S, 1984, *Optimization by simulated annealing: Quantitative studies*, Journal of Statistical Physics, **34(5-6)**, pp. 975–986.

[110] Kolatch E, 2001, *Clustering algorithms for spatial databases: A survey*, (Unpublished) Technical Report CMSC 725, Computer Science Department, University of Maryland, College Park (MD).

[111] Kotsiantis S, Kanellopoulos D & Pintelas P, 2006, *Data preprocessing for supervised leaning*, International Journal of Computer Science, **1(2)**, pp. 111–117.

[112] Kovács F, Legány C & Babos A, 2005, *Cluster validity measurement techniques*, Proceedings of the 6th International Symposium of Hungarian Researchers on Computational Intelligence, Madrid, pp. 388–393.

[113] Kröger B, 1995, *Guillotineable bin packing: A genetic approach*, European Journal of Operational Research, **84(3)**, pp. 645–661.

[114] Lai K & Chan JW, 1997, *Developing a simulated annealing algorithm for the cutting stock problem*, Computers and Industrial Engineering, **32(1)**, pp. 115–127.

[115] Leung SC & Zhang D, 2011, *A fast layer-based heuristic for non-guillotine strip packing*, Expert Systems with Applications, **38(10)**, pp. 13032–13042.

[116] Leung SC, Zhang D & Sim KM, 2011, *A two-stage intelligent search algorithm for the two-dimensional strip packing problem*, European Journal of Operational Research, **215(1)**, pp. 57–69.

[117] Lijun W & Wenbin Z, 2011, *Skyline heuristic for the 2D rectangular packing and strip packing problems, under "Data sets"*, [Online], [Cited July 2017], Available from `https://www.computational-logistics.org/orlib/topic/2D%20Strip%20Packing/index.html`.

[118] Liu D & Teng H, 1999, *An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles*, European Journal of Operational Research, **112(2)**, pp. 413–420.

[119] Lodi A, Martello S & Monaci M, 2002, *Two-dimensional packing problems: A survey*, European Journal of Operational Research, **141(2)**, pp. 241–252.

[120] Lodi A, Martello S & Vigo D, 1999, *Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems*, INFORMS Journal on Computing, **11(4)**, pp. 345–357.

[121] Lodi A, Martello S & Vigo D, 2002, *Recent advances on two-dimensional bin packing problems*, Discrete Applied Mathematics, **123(1)**, pp. 379–396.

[122] Martello S, Monaci M & Vigo D, 2003, *An exact approach to the strip-packing problem*, INFORMS Journal on Computing, **15(3)**, pp. 310–319.

[123] Martello S & Vigo D, 1998, *Exact solution of the two-dimensional finite bin packing problem*, Management Science, **44(3)**, pp. 388–399.

[124] Michalski RS & Stepp RE, 1983, *Automated construction of classifications: Conceptual clustering versus numerical taxonomy*, IEEE Transactions on Pattern Analysis and Machine Intelligence, **(4)**, pp. 396–410.

[125] Milligan GW & Cooper MC, 1985, *An examination of procedures for determining the number of clusters in a data set*, Psychometrika, **50(2)**, pp. 159–179.

[126] Murtagh F, 1983, *A survey of recent advances in hierarchical clustering algorithms*, The Computer Journal, **26(4)**, pp. 354–359.

[127] Nikolaos K, 2014, *Critical values for the Nemenyi test*, [Online], [Cited November 2017], Available from `http://kourentzes.com/forecasting/2014/05/01/critical-values-for-the-nemenyi-test/`.

[128] Ntene N, 2007, *An algorithmic approach to the 2D oriented strip packing problem*, PhD Dissertation, Stellenbosch University, Stellenbosch.

[129] Oehler KL & Gray RM, 1995, *Combining image compression and classification using vector quantization*, IEEE Transactions on Pattern Analysis and Machine Intelligence, **17(5)**, pp. 461–473.

[130] Oliver IM, Smith DJ & Holland JRC, 1987, *A study of permutation crossover operators on the traveling salesman problem*, Proceedings of the 2^{nd} International Conference on Genetic Algorithms and their Application, Cambridge (MA), pp. 224–230.

[131] Ortmann FG, 2010, *Heuristics for offline rectangular packing problems*, PhD Dissertation, Stellenbosch University, Stellenbosch.

[132] Panigrahy R, Talwar K, Uyeda L & Wieder U, 2011, *Heuristics for vector bin packing*, [Online], [Cited June 2016], Available from `http://research.microsoft.com/pubs/147927/VBPackingESA11.pdf`.

[133] Pinto E & Oliveira JF, 2005, *Algorithm based on graphs for the non-guillotinable two-dimensional packing problem*, Proceedings of the 2^{nd} EURO Special Interest Group on Cutting and Packing (ESICUP) Meeting, Southampton, no page numbers.

[134] Pisinger D, 2002, *Heuristics for the container loading problem*, European Journal of Operational Research, **141(2)**, pp. 382–392.

[135] Pisinger D & Sigurd M, 2005, *The two-dimensional bin packing problem with variable bin sizes and costs*, Discrete Optimization, **2(2)**, pp. 154–167.

[136] Pisinger D & Sigurd M, 2007, *Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem*, INFORMS Journal on Computing, **19(1)**, pp. 36–51.

[137] Pureza V & Morabito R, 2006, *Some experiments with a simple tabu search algorithm for the manufacturer's pallet loading problem*, Computers and Operations Research, **33(3)**, pp. 804–819.

[138] Ratanapan K & Dagli CH, 1997, *An object-based evolutionary algorithm for solving rectangular piece nesting problems*, Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Orlando (FL), pp. 989–994.

[139] Rousseeuw PJ, 1987, *Silhouettes: A graphical aid to the interpretation and validation of cluster analysis*, Journal of Computational and Applied Mathematics, **20**, pp. 53–65.

[140] Savić A, Šukilović T & Filipović V, 2011, *Solving the two-dimensional packing problem with mM calculus*, Yugoslav Journal of Operations Research, **21(1)**, pp. 93–102.

[141] Sawant K, 2014, *Adaptive methods for determining DBSCAN parameters*, International Journal of Innovative Science, Engineering and Technology, **1(4)**, pp. 329–334.

[142] Scheithauer G, 1999, *LP-based bounds for the container and multi-container loading problem*, International Transactions in Operational Research, **6(2)**, pp. 199–213.

[143] Silva E, Oliveira JF & Wäscher G, 2014, *2DCPackGen: A problem generator for two-dimensional rectangular cutting and packing problems*, European Journal of Operational Research, **237(3)**, pp. 846–856.

[144] Sleator DD, 1980, *A 2.5 times optimal algorithm for packing in two dimensions*, Information Processing Letters, **10(1)**, pp. 37–40.

[145] Sneath PH & Sokal RR, 1975, *Numerical taxonomy: The principles and practice of numerical classification*, The Quarterly Review of Biology, **50(4)**, pp. 525–526.

[146] Soni N & Kumar T, 2014, *Study of various mutation operators in genetic algorithms*, International Journal of Computer Science and Information Technologies, **5(3)**, pp. 4519–4521.

[147] Syswerda G, 1991, *A study of reproduction in generational and steady-state genetic algorithms*, pp. 94–101 in Banzhaf W (Ed), *Foundations of genetic algorithms*, Elsevier, San Mateo (CA).

[148] Syswerda G, 1991, *Schedule optimisation using genetic algorithms*, pp. 332–349 in Davis L (Ed), *Handbook of genetic algorithms*, Van Nostrand Reinhold, New York (NY).

[149] Theodoridis S & Koutroumbas K, 2008, *Pattern recognition*, IEEE Transactions on Neural Networks, **19(2)**, pp. 376.

[150] Tibshirani R, Walther G & Hastie T, 2001, *Estimating the number of clusters in a data set via the gap statistic*, Journal of the Royal Statistical Society: Series B (Statistical Methodology), **63(2)**, pp. 411–423.

[151] Valenzuela CL & Wang PY, 2001, *Heuristics for large strip packing problems with guillotine patterns: An empirical study*, Proceedings of the 4th Metaheuristics International Conference, Porto, pp. 417–421.

[152] Van Laarhoven PJ & Aarts EH, 1987, *Simulated annealing*, pp. 7–15 in Van Laarhoven PJ & Aarts EH (Eds), *Simulated annealing: Theory and applications*, Springer, Dordrecht.

[153] Van Vuuren JH & Ortmann FG, 2010, *Benchmarks*, [Online], [Cited July 2017], Available from `http://www.vuuren.co.za/main.php`.

[154] Vessel Finder, 2016, *Shanghai Express — Container ship*, [Online], [Cited December 2017], Available from `https://www.vesselfinder.com/ship-photos/185867?s=1`.

[155] Von Luxburg U, 2007, *A tutorial on spectral clustering*, Statistics and Computing, **17(4)**, pp. 395–416.

[156] Wang PY & Valenzuela CL, 2001, *Data set generation for rectangular placement problems*, European Journal of Operational Research, **134(2)**, pp. 378–391.

[157] Wäscher G, Haussner H & Schumann H, 2007, *An improved typology of cutting and packing problems*, European Journal of Operational Research, **183(3)**, pp. 1109–1130.

[158] Wei L, Qin H, Cheang B & Xu X, 2016, *An efficient intelligent search algorithm for the two-dimensional rectangular strip packing problem*, International Transactions in Operational Research, **23(1-2)**, pp. 65–92.

[159] Whitley LD, 1989, *The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best*, Proceedings of the 3rd International Conference on Genetic Algorithms, Fairfax County (VA), pp. 116–123.

[160] Wold S, Esbensen K & Geladi P, 1987, *Principal component analysis*, Chemometrics and Intelligent Laboratory Systems, **2(1-3)**, pp. 37–52.

[161] Xu R & Wunsch D, 2005, *Survey of clustering algorithms*, IEEE Transactions on Neural Networks, **16(3)**, pp. 645–678.

[162] Yang S, Han S & Ye W, 2013, *A simple randomized algorithm for two-dimensional strip packing*, Computers and Operations Research, **40(1)**, pp. 1–8.

[163] Zelnik-Manor L & Perona P, 2005, *Self-tuning spectral clustering*, Proceedings of the Advances in Neural Information Processing Systems, Vancouver, pp. 1601–1608.

[164] Zhang DF, Sheng-Da C & Yan-Juan L, 2007, *An improved heuristic recursive strategy based on genetic algorithm for the strip rectangular packing problem*, Acta Automatica Sinica, **33(9)**, pp. 911–916.

[165] Zhang D, Kang Y & Deng A, 2006, *A new heuristic recursive algorithm for the strip rectangular packing problem*, Computers and Operations Research, **33(8)**, pp. 2209–2217.