

---

# A Decision Support System for the Two-Dimensional Strip Packing Problem

---



UNIVERSITY OF CAPE TOWN

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

HONOURS IN STATISTICS

*in the*

DEPARTMENT OF STATISTICAL SCIENCES

**Author:**

Aidan Wallace

Leslie Wu

**Supervisor:**

Dr. R. Georgina

Rakotonirainy

Submitted: 2 December 2020

# Declaration of Authorship

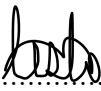
We, Aidan Wallace and Leslie Wu, declare that this project titled, “A Decision Support System for the Two-Dimensional Strip Packing Problem” and the work presented in it is our own.

We confirm that:

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

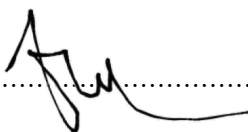
Name: Leslie Wu

Date: 2 December 2020

Signed: 

Name: Aidan Wallace

Date: 2 December 2020

Signed: 

University of Cape Town

## **Abstract**

Faculty of Science, Faculty of Commerce  
Department of Statistical Sciences

### **A Decision Support System for the Two-Dimensional Strip Packing Problem**

Aidan Wallace & Leslie Wu

Under the supervision of  
Dr. R. Georgina Rakotonirainy.

The two-dimensional strip packing problem consists of packing a set of rectangular items into a single object of fixed width in a non-overlapping manner, with the objective of minimising its height. This problem has a wide range of applications, and is typically encountered in the wood, glass and paper industries. Due to the complexity and combinatorial nature of the problem, the development of fast and efficient packing algorithms, mainly employing heuristics and metaheuristic techniques, has been the major concern of many researchers in the field. The aim of this project is twofold. The first aim is to conduct a literature survey on both the seminal and most recent solution approaches for the strip packing problem. The second aim is to design a user-friendly computerised decision support system capable of solving strip packing problems of which the working is based on the solution methodology reviewed in the first aim.

# Acknowledgements

Thank you to Dr. R. Georgina Rakotonirainy, whose generous help and guidance has been of immeasurable value. This past year has been challenging and so for your patience and support we are incredibly grateful. Many thanks, as well, for providing us with the resources to fast track our progress.

Leslie Wu would also like to acknowledge the financial assistance of the National Research Foundation (NRF) towards funding his study at the University of Cape Town.

---

# Table of Contents

<b>Declaration of Authorship</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>List of Acronyms</b>	<b>vii</b>
<b>List of Tables</b>	<b>viii</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Algorithms</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Description . . . . .	1
1.2 Research Aim and Objectives . . . . .	3
1.3 Report Structure . . . . .	4
<b>I Literature Review</b>	<b>6</b>
<b>2 Heuristic Algorithms</b>	<b>7</b>
2.1 Best-Fit Algorithm . . . . .	9
2.1.1 The Burke algorithm . . . . .	9

---

2.1.2	The Wei Algorithm . . . . .	11
2.2	Constructive Heuristic . . . . .	17
2.3	Chapter Summary . . . . .	23
<b>3</b>	<b>Metaheuristic Algorithms</b>	<b>24</b>
3.1	Hybrid Genetic Algorithms . . . . .	27
3.2	Hybrid Simulated Annealing . . . . .	33
3.3	Chapter Summary . . . . .	39
<b>II</b>	<b>Decision Support System</b>	<b>41</b>
<b>4</b>	<b>System Development</b>	<b>42</b>
4.1	The 2D Strip Packing DSS . . . . .	43
4.2	Development and Implementation of the DSS . . . . .	45
4.3	Chapter Summary . . . . .	48
<b>5</b>	<b>System Interface</b>	<b>50</b>
5.1	Tutorial . . . . .	50
5.2	Instances . . . . .	52
5.2.1	Benchmark Instances . . . . .	52
5.2.2	User Instances . . . . .	53
5.2.3	Instances Summary . . . . .	53
5.3	Algorithms . . . . .	53
5.3.1	Heuristics . . . . .	54
5.3.2	Metaheuristics . . . . .	55
5.3.3	Output . . . . .	57
5.4	Comparison . . . . .	58
5.5	About . . . . .	59

---

5.6 Chapter Summary . . . . .	60
<b>III Conclusion</b>	<b>62</b>
<b>6 Dissertation Summary</b>	<b>63</b>
6.1 Summary of Dissertation Contents . . . . .	63
<b>7 Future Work</b>	<b>65</b>
7.1 Alternative SPP Solution Techniques . . . . .	65
7.2 Improvements to the DSS . . . . .	69
<b>A Benchmark Instances</b>	<b>70</b>
<b>References</b>	<b>74</b>

---

# List of Acronyms

**BF:** Best-Fit

**BL:** Bottom-Left

**CH:** Constructive Heuristic

**C&P:** Cutting and Packing

**DSS:** Decision Support System

**GA:** Genetic Algorithm

**IBF:** Improved Best-Fit

**ICH:** Improved Constructive Heuristic

**OX:** Order Crossover

**PMX:** Partially Matched Crossover

**SA:** Simulated Annealing

**SPP:** Strip Packing Problem

**2D SPP:** Two-Dimensional Strip Packing Problem



---

## List of Tables

2.1	Dimensions of the rectangles in $\mathcal{H}$ used as an example instance . . . . .	9
2.2	The scoring rules employed in the CH algorithm . . . . .	18
2.3	Packing heights of the heuristics applied to the items in $\mathcal{H}$ for different initialised orders . . . . .	23
3.1	Dimensions of the rectangles in $\mathcal{I}$ used as an example instance . . . . .	26
7.1	The scoring rules employed in the ICH algorithm . . . . .	67
7.2	Performance of the proposed ICH relative to the CH . . . . .	68

---

# List of Figures

1.1	Examples of cutting and packing problems in real-world applications . . . . .	1
1.2	Example instance of a 2D SPP . . . . .	2
2.1	Examples of level, pseudolevel, and plane packing solutions . . . . .	8
2.2	The item set $\mathcal{H}$ used for illustrative purposes . . . . .	9
2.3	The tallest neighbour and shortest neighbour placement policies . . . . .	10
2.4	Example of a skyline . . . . .	12
2.5	Example of the scoring rule employed in the IBF algorithm . . . . .	14
2.6	Items in $\mathcal{H}$ packed by means of the IBF algorithm . . . . .	16
2.7	Items in $\mathcal{H}$ , sorted by decreasing height, packed by means of the IBF algorithm . . . . .	16
2.8	Items in $\mathcal{H}$ , sorted by decreasing width, packed by means of the IBF algorithm . . . . .	17
2.9	Example of the scoring rule employed in the CH algorithm . . . . .	19
2.10	Items in $\mathcal{H}$ packed by means of the CH algorithm . . . . .	20
2.11	Items in $\mathcal{H}$ , sorted by decreasing height, packed by means of the CH algorithm . . . . .	21

2.12	Items in $\mathcal{H}$ , sorted by decreasing width, packed by means of the CH algorithm . . . . .	21
3.1	The item set $\mathcal{I}$ used for illustrative purposes . . . . .	27
3.2	Example of Order Crossover . . . . .	30
3.3	Example of Partially Matched Crossover . . . . .	31
3.4	Example of swap mutation . . . . .	31
3.5	A comparison of the IBF algorithm and the hybrid GA with IBF applied to pack $\mathcal{I}$ . . . . .	33
3.6	Temperature profile of the cooling schedules . . . . .	36
3.7	Temperature profile of the cooling schedules with recommended rate parameters . . . . .	37
3.8	A comparison of the CH algorithm and the hybrid SA with CH applied to pack $\mathcal{I}$ . . . . .	39
4.1	<b>packR</b> folder directory tree structure diagram . . . . .	44
4.2	A screen shot of an Excel file specifying a problem instance . . . . .	46
4.3	Flow diagram of the DSS components and structure . . . . .	49
5.1	Screen shots of the Tutorial pop-ups of the DSS . . . . .	51
5.2	Screen shot of the Instances tab of the DSS . . . . .	52
5.3	Screen shot of the Algorithms tab of the DSS . . . . .	54
5.4	GA parameters in the DSS. . . . .	55
5.5	SA parameters in the DSS. . . . .	56
5.6	Example metaheuristic output of the DSS in the Heuristics tab . . . . .	58
5.7	Screen shot of the Compare tab of the DSS . . . . .	59
5.8	Packing layout plot generated in the DSS . . . . .	60
5.9	Screen shot of the About tab of the DSS . . . . .	61

---

7.1 The items of $\mathcal{H}$ and $\mathcal{I}$ packed with the ICH scoring . . . . .	68
--	----

---

# List of Algorithms

1	Best-Fit Heuristic . . . . .	11
2	Improved Best-Fit Heuristic . . . . .	13
3	Constructive Heuristic . . . . .	22
4	Hybrid Genetic Algorithm . . . . .	32
5	Hybrid Simulated Annealing Algorithm . . . . .	38

---

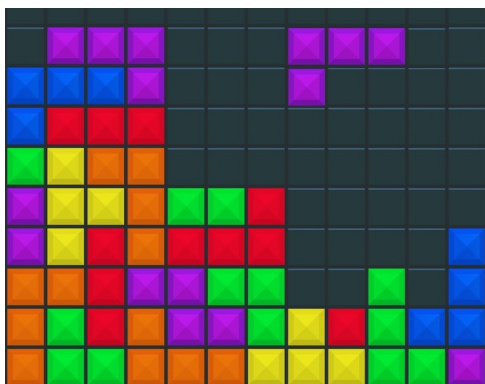
# CHAPTER 1

---

## Introduction

### 1.1 Problem Description

Cutting and packing (C&P) problems cover a general class of complex combinatorial optimisation problems. These problems are rooted in finding optimal strategies in which small items are required to be cut from or packed into larger objects to minimise waste (unused regions of the large object) [74]. C&P problems have a broad range of applications in areas such as management sciences, operations research, mathematical sciences, and engineering [28]. The ability to reduce raw material waste during cutting or packing procedures is linked to a reduction in cost in various industrial contexts. Despite the study of these problems arising largely from problems found in industry, C&P problems are widely encountered in everyday settings; take for instance playing the famous and recognizable Tetris video game, or even finding the optimal arrangement of fruits and vegetables to be placed on a store shelf (see Figure 1.1). Additionally, C&P problems also enter as a facet of other complex problems, and these include, but not limited to, routing problems with capacity and scheduling problems with resource constraints [26].



(a) Tetris, a tile-matching video game [92].



(b) Packing of oranges [65].

Figure 1.1: Examples of C&P problems in practice.

The *Two-Dimensional Strip Packing Problem* (2D SPP) is fundamental to the class of C&P problems. The 2D SPP is concerned with finding a suitable arrangement of a set of  $n$  two-dimensional rectangular objects with dimensions  $w_i \times h_i$  for  $i = 1, \dots, n$ , into an area (strip) of fixed width and unlimited height in a non-overlapping manner, such that the height of the packed objects is minimised. An illustration of such a problem in which seven rectangles ( $n = 7$ ) are packed is presented in Figure 1.2. This example of a 2D SPP falls into the class of offline strip packing problems. Offline problems are those where the entire set of items to be packed is known in advance. These problems are in contrast to the on-line packing problems, in which the entire set of rectangles is not known in advance. It is the former set of problems, offline problems, and the suite of techniques used to solve these problems that frame the focus of this project.

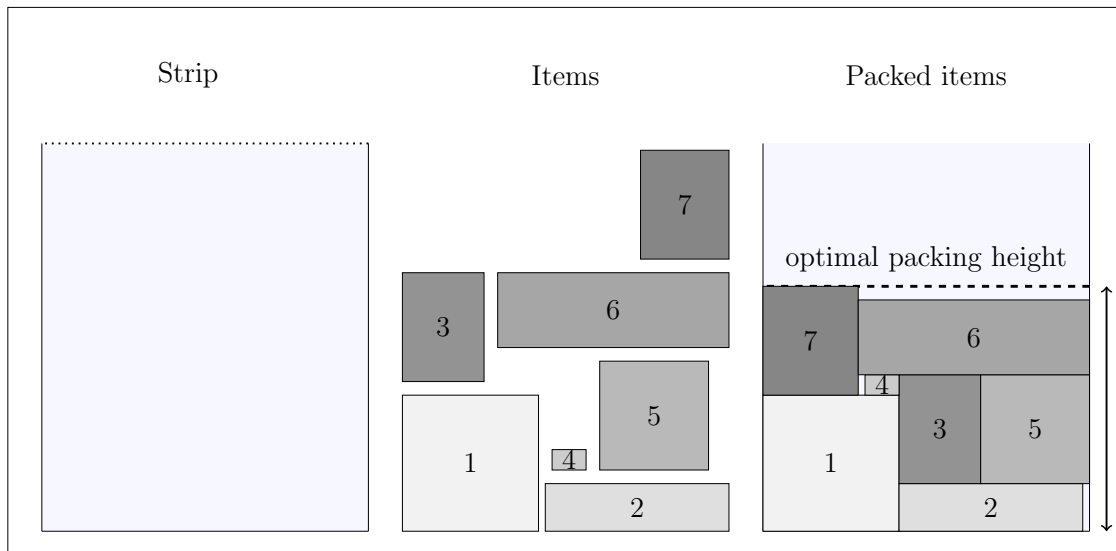


Figure 1.2: Example of a two-dimensional strip packing problem.

These problems have widespread practical relevance in industries such as logistics (e.g., cargo packing) and manufacturing (e.g., trim loss) where resource efficiency is essential. In these applications, the goals of each activity are directly related; to produce good quality arrangements of items so as to maximise material utilisation is equivalent to minimising wastage. This has in turn led to spurring interest and the incentivising of research in the field.

Exact mathematical methods, often also referred to as deterministic methods, have been proposed for the 2D SPP, and whose formulations are rooted in mathematical programming modelling. Methods as such include, but are not limited to, Gilmore and Gomory's *column generation method* [32, 33, 34], Christofides and Whitlock's *tree-search algorithm* [19], and Martello and Vigo's *branch-and-bound algorithm* [62]. However the nature of the 2D SPPs is rooted in the class of NP-hard [44]. In fact, these problems are a generalisation of the cutting stock problem, which in turn is a generalisation of the famous

knapsack problem [39]. The running times for any of these algorithms currently known to guarantee an optimal solution are an exponential function of the size of the problem. The exact algorithms of Hifi [39], Kenmochi *et al.* [48], Boschetti and Montaletti [12], and Belov [7], for instance, all face the shortcoming that they can only be applied to solve small size instances. Exact algorithms, in short, are impractical for solving large-scale problems due to computational infeasibility [29]. Nevertheless, these methods are useful in guaranteeing an optimal solution, or at least providing an indication of solution quality.

Solution quality and computational times are important considerations for practitioners. In industries such as logistics, computational time and the urgency of a solution may be deemed more important than solution quality, whereas in other applications where material waste is expensive, solution quality may be more important than computational time. Often the trade-off of a marginal decrease in solution quality with algorithm efficiency is sufficient and well worthwhile, and vice versa. It is to this extent that approximate techniques have been designed, studied, and utilised in practice.

## 1.2 Research Aim and Objectives

The literature on strip packing is vast, and this problem has been widely studied over the years. Numerous approaches to the strip packing problem have been proposed in the literature, and these fast and efficient packing algorithms fall into the class of (1) exact methods; (2) approximate or heuristics methods; or (3) hybrid or metaheuristic methods [47].

The focus of this project is divided into two main parts, namely:

- ▶ a literature review of both seminal and most recent solution approaches of the classes (2) and (3) aforementioned for the strip packing problem; and
- ▶ the creation of a decision support system (DSS) for solving two-dimensional strip packing problems.

Firstly, we will conduct an extensive review of past research in the area of 2D SPPs and provide a general overview of the different techniques that have been applied previously. This study will be centered on considering several proposed heuristic and metaheuristic techniques for solving offline strip packing problems approximately. At this stage, we clarify precisely the specifications of the problems considered in this project – we consider the orthogonal fixed orientation variant of the 2D SPP in which the rotation of the rectangles is not allowed in the process of packing. Furthermore, we do not account for guillotine cuts in the packing procedure.

Part two will consist of the development of a user-friendly computerised decision support system for the 2D SPP. The goal of such a decision support system is to provide users



with an easy and accessible way of solving strip packing problems of which the working is based on the solution methodology reviewed in the first aim.

The following objectives outline a clear path to realising the aims of this project:

- ▶ *conduct* a thorough literature review of the various proposed heuristic and meta-heuristic strategies;
- ▶ *select* a subset of heuristic and metaheuristic approaches that will be implemented in the decision support system;
- ▶ *identify* problem solutions that will be applicable for the decision support system;
- ▶ *implement* the algorithms within the framework in a software based decision support system; and
- ▶ *provide* a variety of evaluation measures and adequate sensitivity analysis of the generated solutions for comparison of algorithms based on the user's problem instance.

## 1.3 Report Structure

This research project is organised into six chapters following this current introductory chapter divided over three parts. The first part, consisting of two chapters, is dedicated to reviewing the literature on existing algorithmic approaches to generating solutions to the 2D SPP. The second part of two chapters focuses on the implementation of the DSS for solving instances of the 2D SPP based on the work contained in the part previous. The closing part, which also contains two chapters, is devoted to a summary of this project as well as to lay some of the ground work for future work.

In [Chapter 2](#), a background on heuristic algorithms is presented. The chapter begins with a brief description of heuristic algorithms from the literature before presenting two plane algorithms; this first is by Burke *et al.* [15] with improvements proposed by Wei *et al.* [93], and the second by Leung *et al.* [53]. Plane algorithms are those which pack items anywhere in the region defined by the boundaries of the strip and without any level constraints. This is followed by a look into metaheuristic approaches to the 2D SPP in [Chapter 3](#). The chapter opens with a discussion on the basic notions that have to be borne in mind when using metaheuristic methodologies and the remainder of the chapter thereafter commits to a review of the hybrid genetic algorithm and hybrid simulated annealing solution approach to the SPP.

[Chapter 4](#) introduces a computerised DSS for SPPs. This chapter presents the fundamental principles integral to the development of a DSS as well as outlines and motivates the methodology that was followed in its design and implementation. The interface of

the DSS is presented in [Chapter 5](#) by means of annotated screenshots. This chapter deals with displaying the various components of the DSS and the interplay thereof as one navigates through the software.

A summary of all the main elements of the work presented in this project is provided in [Chapter 6](#). In [Chapter 7](#), suggested avenues to explore for any follow-up investigations are given.

Finally, supplementary material to accompany the work of this project is presented in [Appendix A](#). Here, a listing of a sample of benchmark instances found in the 2D SPP literature and utilised in the course of this project is provided and each accompanied by some insight into their respective construction.

**Part I**  
**Literature Review**

---

# CHAPTER 2

---

## Heuristic Algorithms

The heuristics, and in general the approximate techniques, are designed to produce good, although not necessarily optimal solutions, within a reasonable computing time. The trade-off is that these techniques are not able to evaluate the distance to optimality. This bears the consequence that they are not able to recognise an optimal solution if found.

In studying packing problems, the heuristics are algorithms that work by determining the placement of items in the strip as guided by a set of predefined rules. There are varying methodologies that have been presented for solving the 2D SPP, however the algorithms, for a given packing order, can be classified as falling into the subclasses of *level algorithms*, *pseudolevel algorithms*, and *plane algorithms*. Level algorithms define a collection of horizontal lines drawn across the strip joining the two vertical sides upon which items are packed with the condition that at least one edge of each item packed must lie on the lower boundary of a level. The first level of the strip coincides with the bottom of the strip, and the height of each subsequent level thereafter is determined by the height of the tallest rectangle on the previous level. Pseudolevel algorithms give packing arrangements that are similarly specified by levels but without the necessity of having an edge coinciding with the lower boundary of a level. In both level and pseudolevel algorithms no packed items intersect any level boundary. Plane algorithms allow for packing anywhere in the region defined by the boundaries of the strip; that is, there is no partitioning of the strip into levels. By way of explanation, these heuristic types when applied to solving an example packing instance are provided in Figure 2.1.

Coffman *et al.* [20], Berkey and Wang [9], and Martello *et al.* [61] have contributed towards level-packing heuristics for the SPP, while Lodi *et al.* [57, 58], Bortfeldt [10], Ntene and Van Vuuren [67], and Ortmann *et al.* [68] have proposed pseudolevel packing heuristics for the same problem. In this chapter, our focus is on considering algorithms belonging to the class of plane algorithms.

The work of Sleator [86] is one of the earliest known records of a proposed plane heuristic for the 2D SPP. The algorithm stacks all items with width exceeding half the width

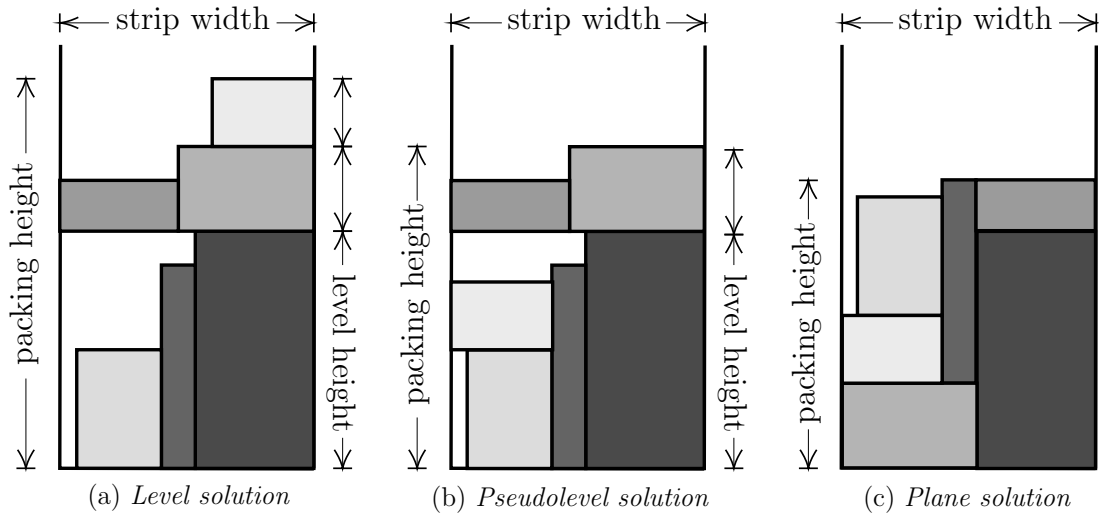


Figure 2.1: An illustration of strip packing solutions generated using level, pseudolevel and plane algorithms as applied to the same problem instance.

of the strip from bottom up. The remaining items are sorted in non-increasing order and are packed on the last packed item left to right until there is insufficient space or there are no remaining unpacked items. At this point the strip is partitioned into two equal halves. The heights of these two left and right strip-segments coincide with the height of the top edge of the tallest item in each respective segment and this defines left and right baselines. The algorithm proceeds to pack the unpacked rectangles in the strip-segment with the lower baseline from left to right in a next-fit fashion. This process is repeatedly continued until no further items remain to be packed.

Two proposed heuristic algorithms in the literature for optimising the 2D SPP that are plane algorithms will be examined in detail in what follows, namely, the *best-fit heuristic* and the *constructive heuristic*. The study in to the former is rooted in the work of Burke *et al.* [15] and which has been further expanded upon and improved in recent times by Wei *et al.* [93], while the latter is based on the proposed packing routine of Leung *et al.* [53]. An explanation of each algorithm is presented, and followed by an implementation in pseudocode together with a worked example for illustrative purposes. There are, however, numerous other plane-packing algorithms, including but not limited to the *split-fit* algorithm of Coffman *et al.* [20], the *bottom-up left-justified* algorithms proposed by Baker *et al.* [5], and Golan's *split* algorithm and *mixed* algorithms [36].

The problem instance proposed and studied by Burke *et al.* in their paper [15] will be used extensively as a running example to demonstrate the workings of the algorithms presented in this chapter. In the paper, twelve instances are presented and it is, particularly, the first of these proposed instances that will be considered. The items in this instance that are to be packed will be referred to by the set  $\mathcal{H}$ . The construction of

these instances was done in accordance with their proposed benchmark problem instance generator in the studied paper. Briefly, the algorithm describes a routine to divide an initial large rectangle into a desired number of smaller sub-rectangles all of which satisfy specified dimension constraints. The division is done by making repeated random horizontal and vertical cuts to randomly selected rectangles. For this particular problem instance under consideration, there are 10 items to be packed into a strip of width 40 with the optimal packing arrangement giving a resultant height of 40. This instance, moreover the set of problem instances, was obtained from [89]. The dimensions (height, width) of each of the items contained in the set  $\mathcal{H}$  are presented in Table 2.1, and an illustration of these items can be found in Figure 2.2.

Table 2.1: Dimensions  $(h(\mathcal{H}_i), w(\mathcal{H}_i))$  of the rectangles in  $\mathcal{H}$  used as an example instance.

Items, $\mathcal{H}_i$	1	2	3	4	5	6	7	8	9	10
Height, $h(\mathcal{H}_i)$	6	16	20	24	4	4	8	20	4	6
Width, $w(\mathcal{H}_i)$	7	40	5	24	7	4	7	4	5	7

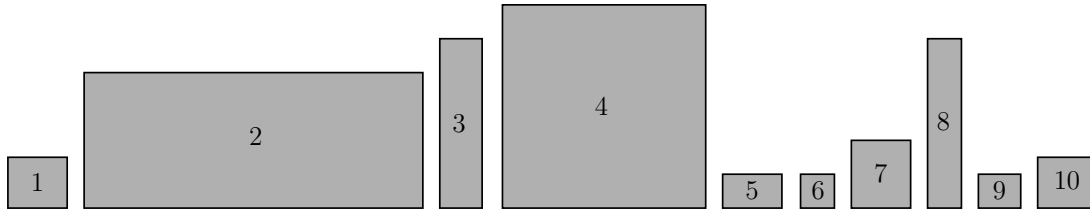


Figure 2.2: A scaled graphical representation of the items in  $\mathcal{H}$ .

## 2.1 Best-Fit Algorithm

### 2.1.1 The Burke algorithm

Numerous strip packing heuristics follow a two-stage procedure in the packing of a list of unpacked rectangles, and, in broad terms, the basic steps consist of either making a rectangle selection – the first unpacked one – and then finding the most suitable position for packing, or choosing a position first and thereafter finding the best-fit rectangle. The algorithm of Burke *et al.* [15] is of the second type and is referred to as the *best-fit* (BF) algorithm in the literature. This algorithm was proposed to solve the non-guillotineable SPP in which 90 degree rotations of the rectangles are permitted. The notion of rotation in the SPP context is beyond the scope of this project and so only the fixed orientation variant, in which the rotation of the rectangles is not allowed, of the problem is considered and so this algorithm is described here for problems in which rotations are disallowed.

In this algorithm, the step of finding the position for packing coincides with finding the lowest available space (gap) where a rectangle may be placed (ties are resolved by selecting the leftmost gap), while the rectangle of best-fit is determined by dynamically scanning the entire list of unpacked items and making a selection via a priority rule. The methodology of Burke *et al.* is that the rectangle with the largest width fitting the gap is the best rectangle to place in the position. If there are ties then the resolution thereof is achieved using the largest height or area. This two-step procedure is conducted before each rectangle is packed. The idea of a best-fit strategy is, essentially, a greedy algorithm that tries to generate good-quality packing solutions by examining the lowest available gap and placing the rectangle that fits best.

At each iteration in the packing three possibilities can occur: (i) there is a shape with width exactly fitting the gap, (ii) there is a shape with width smaller than the gap, or (iii) no shapes will fit the gap. The case of the first is trivial for placement, while in the second case three gap-placement policies were introduced to place the best-fit rectangles when the rectangle does not fit into the gap exactly: *place at leftmost*, *place next to tallest neighbour*, and *place next to shortest neighbour*. An example of the placement next to tallest neighbour and placement next to shortest neighbour is shown in Figure 2.3. The algorithm aims never to purposely create gaps, but if the circumstance invalidates any placement in the gap then there exists no choice but to create a new gap. If a gap is found in which none of the unpacked rectangles can be accommodated, then this gap is raised to the height of its shortest neighbour and the space becomes wasted space. This point is sensible in that if this space cannot be filled at this stage by any of the remaining rectangles then at any later iteration this fact remains unchanged. This entire process is repeated until no remaining rectangles are left to pack.

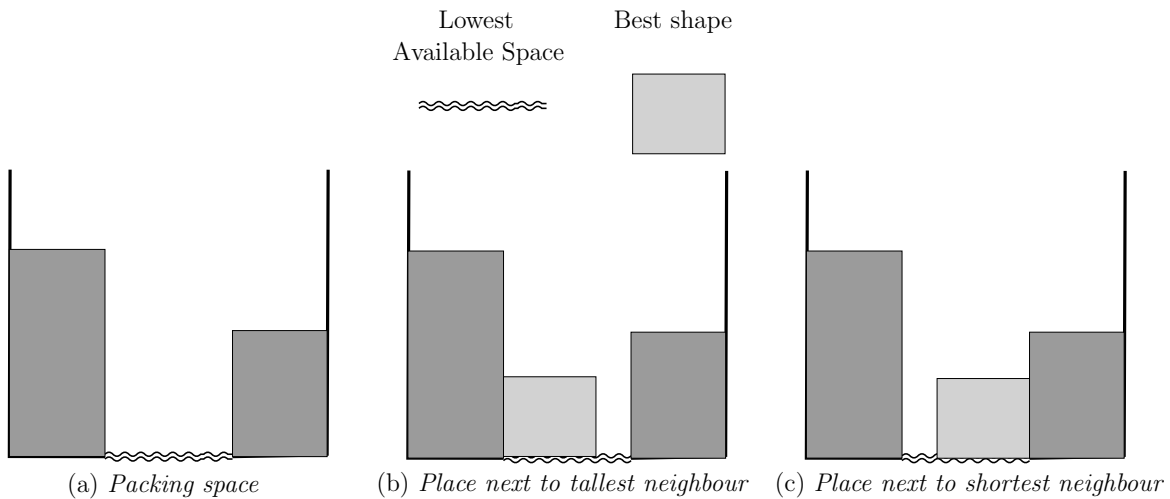


Figure 2.3: Placement of the best-fitting rectangle in the lowest available space according to the *tallest neighbour* and *shortest neighbour* placement policies of the BF algorithm.

In the implementation of the algorithm the concept of a *skyline* or space array, proposed by Burke *et al.* [15], is used to represent and keep track of the set of all available gaps in which the packing of items can be considered. This is updated dynamically as items are placed in the iterative packing procedure. The use of a skyline representation has been used by numerous researches in subsequent work [53, 94, 95, 98]. This notion of a skyline will be made clearer in the ensuing section.

Wei *et al.* [93] note that for the best-fit algorithm the results are independent of the input rectangles sequence and only three solutions are possible and are generated using the different packing placement policies. Usually this algorithm is run for each of the placement methods and the one resulting in the lowest height in the packing layout in the strip is selected. A pseudocode representation of the BF algorithm is outlined in Algorithm 1.

---

**Algorithm 1:** Best-Fit Heuristic

---

**Input** : A list of  $n$  items to be packed,  $\mathcal{R}$ ; the dimensions of the items,  $\langle width[i], length[i] \rangle$ ; and the strip width  $W$ .

**Output** : A feasible packing arrangement of the items into a strip of width  $W$ , and the height of the packed objects.

```

1 initialise the space array  $S$ ;
2 for each placement policy do
3   while there are unpacked items do
4     find the lowest and leftmost space  $s$ ;
5     if an item in  $\mathcal{R}$  fits into  $s$  then
6       find the best-fitting rectangle  $r$  and place using the placement policy;
7       update space array  $S$  by raising elements of array to appropriate height;
8       remove  $r$  from  $\mathcal{R}$ ;
9     else
10      update space array  $S$  by raising space to height of the lowest neighbour;
11    end
12  end
13 end
14 return the best solution when comparing the total packing heights obtained by each
    placement policy;

```

---

## 2.1.2 The Wei Algorithm

Using the similar framework of the BF heuristic, the subsequent work of Imahori and Yagiura [45] gave an efficient implementation of the algorithm, while Leung *et al.* introduced a scoring rule which comprises of five cases to describe a measure of fitness of each rectangle [53]. Yang *et al.* [98] later provided revisions to this scoring



scheme to consider eight cases and this was even further developed by Wei *et al.* [95]. It is this more recent work of Wei *et al.* that we review and describe an improved best-fit approach for the 2D SPP which we will aptly refer to as the *improved best-fit* (IBF) algorithm. As before, this algorithm makes extensive use of the concept of a skyline.

The main difference of the IBF with the BF is due to the introduction of a scoring rule to assess fitness. Where in the BF algorithm the best-fitting rectangle was the one with the largest width fitting in the lowest gap, the IBF employs a scoring rule to assign a fitness to each rectangle in the consideration of packing in the lowest gap in every iteration of the packing. It is noted that the use of a scoring rule, in comparison to the original best-fit measure, is more precise [93]. An additional difference is that the solution obtained by the IBF relates to the input rectangle sequence. This bears the consequence that a different input sequence yields different solutions. The approach of Wei *et al.* classifies only four cases.

Before describing the IBF algorithm and the associated scoring rule, it is important to define the notation used in the discussion. Let  $s_j$  denote each horizontal segment or space in the skyline. Each  $s_j$  is described by the attributes

- $x, y$ : the coordinates of the left corner of  $s_j$ ;
- $\omega$ : the width of  $s_j$
- $h_1$ : the height of the vertical segment at the left-corner of  $s_j$ ; and
- $h_2$ : the height of the vertical segment at the right-corner of  $s_j$ .

An example of a skyline and the attributes of the bottom-leftmost space is given in Figure 2.4.

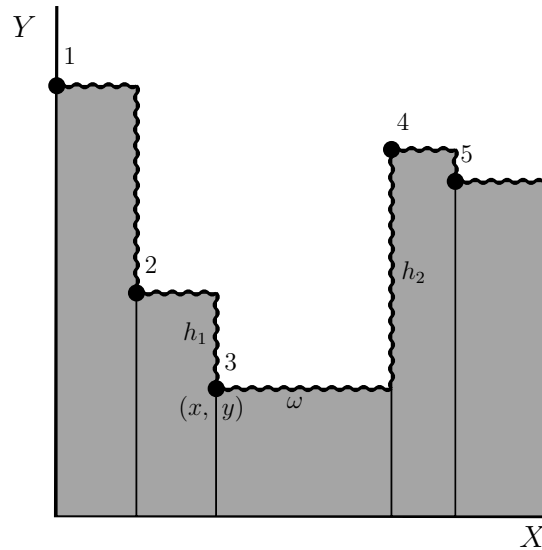


Figure 2.4: Example of a skyline comprising five spaces. The bottom left-most available space is the segment with left endpoint 3.

For  $h_1$  of the first segment  $s_1$  and  $h_2$  of the last segment  $s_k$  these quantities are both set to infinite.

The algorithm for the IBF can be presented in pseudocode as given by [Algorithm 2](#).

---

**Algorithm 2:** Improved Best-Fit Heuristic

---

**Input :** A list of  $n$  items to be packed,  $\mathcal{R}$ ; the dimensions of the items,  $\langle width[i], length[i] \rangle$ ; and the strip width  $W$ .

**Output :** A feasible packing arrangement of the items into a strip of width  $W$ , and the height of the packed objects  $h$ .

---

```

1   $h \leftarrow 0, packedItems \leftarrow 0$ ;
2  initialise the space array  $S$ ;
3  while  $packedItems < n$  do
4      find the lowest and leftmost space  $s = (x, y, \omega, h_1, h_2)$ ;
5      if an item in  $\mathcal{R}$  fits into  $s$  then
6          for each item  $i$  in  $\mathcal{R}$  do
7               $s_i = BestFitScore(i, \omega, h_1, h_2)$ 
8          end
9          Select an item  $r$  such that  $s_r = \max_i \{s_i\}$ ;
10         if  $length[r] + y > h$  then
11              $h \leftarrow length[r] + y$ 
12         end
13         if  $h_1 \geq h_2$  then
14             pack item  $r$  against left wall and remove  $r$  from  $\mathcal{R}$ ;
15              $packedItems \leftarrow packedItems + 1$ ;
16             update space array  $S$ ;
17         else
18             pack item  $r$  against right wall and remove  $r$  from  $\mathcal{R}$ ;
19              $packedItems \leftarrow packedItems + 1$ ;
20             update space array  $S$ ;
21         end
22     else
23         update space array  $S$ ;
24     end
25 end
26 return  $h$ ;

```

---

Line 7 of the algorithm calls for a scoring rule to evaluate the fitness for each unpacked item. For an unpacked rectangle  $r$ , the fitness criterion is one which counts the number of sides of  $r$  that exactly matches the attributes of the bottom-left segment in the skyline. The left side of a rectangle is an exact match if its height is equal to  $h_1$ , and similarly for the right side and  $h_2$ . The bottom side of the rectangle is an exact match if its width

is  $\omega$ . Figure 2.5 demonstrates graphically the scoring rule for evaluating the fitness number for various example rectangles in a space  $s$ .

In the IBF heuristic, the rectangle with the greatest fitness number is selected and the first unpacked rectangle in the sequence of rectangles is used in the case of a tie. Where the BF incorporated a placement policy, the IBF prefers to place a rectangle besides the tallest neighbour if the fitness of both placements concur. For instance, in case (b) of Figure 2.5 placement of the rectangle beside either the left or right side of the space results in a fitness number of 0 and so the placement of this rectangle would be against the tallest neighbour, the right neighbour. Wei *et al.* motivated this idea owing to the reason that it results in a “smoother” skyline with fewer segments which in turn will likely allow for placing larger rectangles without producing wasted space.

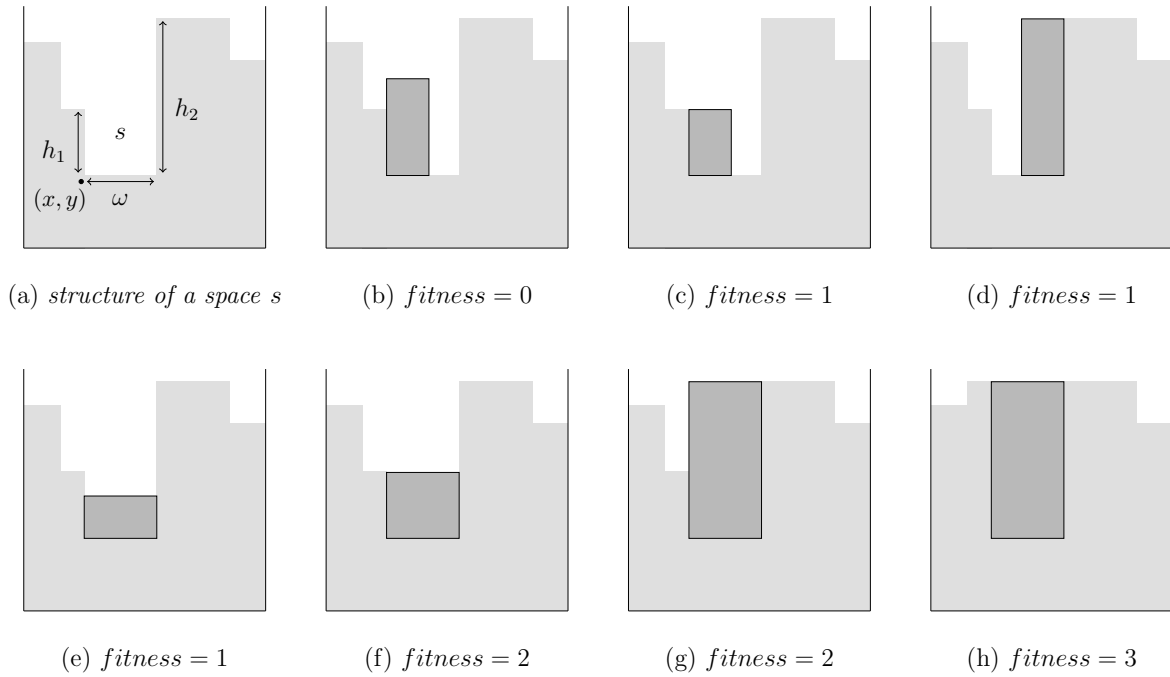


Figure 2.5: Example of the scoring rule in the IBF algorithm (Wei *et al.* [93]).

Application of the IBF algorithm outlined is used for packing the items in the set  $\mathcal{H}$  for different orderings of the input sequence of the rectangles.

**Example 2.1.** The items in the set  $\mathcal{H}$  of Table 2.1 are left in the default order,  $\mathcal{H} = \{\mathcal{H}_1, \dots, \mathcal{H}_{10}\}$ . Initially, the lowest gap is simply the floor of the strip and has width 40. In accordance with the scoring rule, all items except  $\mathcal{H}_2$  has a fitness equal to 0 with respect to the space. The width of  $\mathcal{H}_2$  matches exactly with the width of the space, and so  $\mathcal{H}_2$  has a fitness number 1 and is placed in the space and removed from the list.

The skyline is updated so that the lowest and only space is the segment that spans the top edge of  $\mathcal{H}_2$ . None of the items have a dimension fitting this space exactly, they all have fitness 0, and so the first unpacked item  $\mathcal{H}_1$  is placed into the space left-justified. The number of available spaces for packing is now increased to two: the region above  $\mathcal{H}_1$  and the space spanning the length of  $\mathcal{H}_2$  not under  $\mathcal{H}_1$ . The second space is the lowest left space and a consideration of the remaining unpacked rectangles yields  $\mathcal{H}_{10}$  as the only item with a dimension matching the space exactly – the left side of the rectangle coincides exactly with the height of this space  $h_1$  – and so  $\mathcal{H}_{10}$  is packed beside  $\mathcal{H}_1$  and the skyline is updated once more. The process is similarly continued repeatedly in accordance with the IBF methodology to pack the rectangles  $\mathcal{H}_3, \mathcal{H}_8, \mathcal{H}_5, \mathcal{H}_6, \mathcal{H}_9$ .

At this stage the skyline consists of 4 spaces: the spaces coinciding with the top edge spanned by  $\mathcal{H}_1$  and  $\mathcal{H}_{10}$ , the space above  $\mathcal{H}_3$  and  $\mathcal{H}_8$ , similarly the space above  $\mathcal{H}_5, \mathcal{H}_6$  and  $\mathcal{H}_9$  and the final space given by the region the right side of  $\mathcal{H}_9$  to the right strip boundary sitting above  $\mathcal{H}_2$ . The last of these described 4 spaces is the bottom-leftmost packing space. None of the remaining rectangles, namely,  $\mathcal{H}_4$  and  $\mathcal{H}_7$ , have widths fitting this space and this space is discarded as waste. Hence, the skyline of this space is raised to be the height of the shortest neighbour which in this case coincides with the top edges of  $\mathcal{H}_5, \mathcal{H}_6$  and  $\mathcal{H}_9$  and altogether this becomes the new bottom-leftmost space. Only  $\mathcal{H}_7$  fits this space and thus is packed left-justified.

The item  $\mathcal{H}_4$  has a large width and it is through repeatedly raising the skyline of the bottom-leftmost space iteratively that the bottom-leftmost space becomes the segment length that spans the top of  $\mathcal{H}_1, \mathcal{H}_{10}, \mathcal{H}_3, \mathcal{H}_8, \mathcal{H}_7, \mathcal{H}_6, \mathcal{H}_9$  and  $\mathcal{H}_2$  at the height of the top edges of  $\mathcal{H}_3$  and  $\mathcal{H}_8$  that  $\mathcal{H}_4$  can be packed. The final packing layout is shown in Figure 2.6. This packing arrangement results in a height of 60 which deviates quite widely from the known optimal height of 40.

**Example 2.2.** The items in the set  $\mathcal{H}$  of Table 2.1 are sorted according to decreasing height and resolving ties by additionally sorting these items according to the default ordering yields the list  $\mathcal{H}' = \{\mathcal{H}_4, \mathcal{H}_3, \mathcal{H}_8, \mathcal{H}_2, \mathcal{H}_7, \mathcal{H}_1, \mathcal{H}_{10}, \mathcal{H}_5, \mathcal{H}_6, \mathcal{H}_9\}$ . The packing of the set  $\mathcal{H}'$  is performed similarly as in the example previous with the IBF heuristic. The order of the packing of the rectangles is  $\mathcal{H}_2, \mathcal{H}_4, \mathcal{H}_3, \mathcal{H}_8, \mathcal{H}_7, \mathcal{H}_1, \mathcal{H}_{10}, \mathcal{H}_5, \mathcal{H}_6, \mathcal{H}_9$ . The final packing layout is shown in Figure 2.7. This packing arrangement coincides with the known optimal height of 40.

**Example 2.3.** The items in the set  $\mathcal{H}$  of Table 2.1 are sorted according to decreasing width and resolving ties by additionally sorting these items according to the default ordering yields the list  $\mathcal{H}^* = \{\mathcal{H}_2, \mathcal{H}_4, \mathcal{H}_1, \mathcal{H}_5, \mathcal{H}_7, \mathcal{H}_{10}, \mathcal{H}_3, \mathcal{H}_9, \mathcal{H}_6, \mathcal{H}_8\}$ . The packing of the set  $\mathcal{H}^*$  is performed similarly as in the example previous with the IBF heuristic. The order of the packing of the rectangles is  $\mathcal{H}_2, \mathcal{H}_4, \mathcal{H}_1, \mathcal{H}_{10}, \mathcal{H}_5, \mathcal{H}_9, \mathcal{H}_6, \mathcal{H}_7, \mathcal{H}_3, \mathcal{H}_8$ . The final packing layout is shown in Figure 2.8. The produced packing layout gives rise to a packing height of 46, slightly above the optimal height 40.

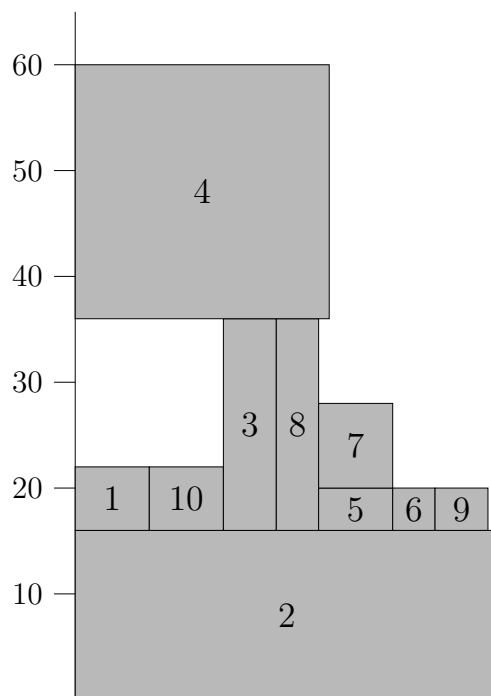


Figure 2.6: Result of packing the items  $\mathcal{H}$  using the IBF algorithm. The items are presented to the algorithm in the default order. The resulting packing height is 60.

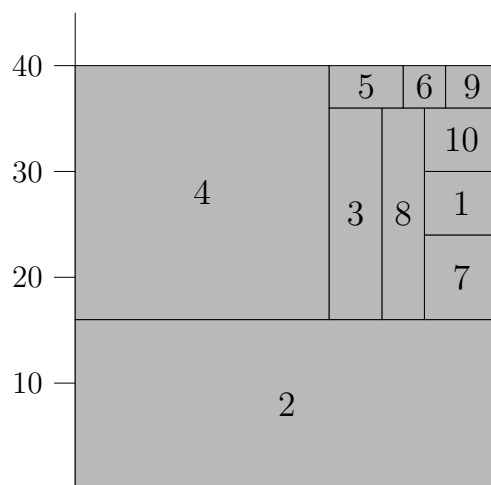


Figure 2.7: Result of packing the items  $\mathcal{H}$  using the IBF algorithm. The items are presented to the algorithm by sorting the items in  $\mathcal{H}$  by decreasing height. The resulting packing height is 40.

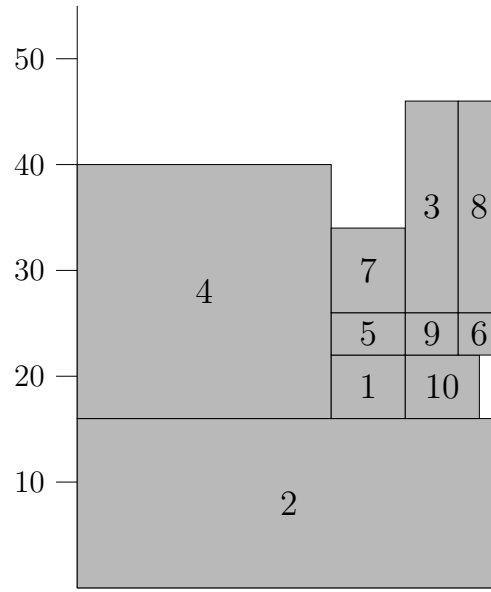


Figure 2.8: Result of packing the items  $\mathcal{H}$  using the IBF algorithm. The items are presented to the algorithm by sorting the items in  $\mathcal{H}$  by decreasing width. The resulting packing height is 46.

## 2.2 Constructive Heuristic

The constructive heuristic proposed by Leung *et al.* [53] is an effective algorithm for providing a quick approximate solution to the orthogonal 2D SPP without rotation, especially for large-scale problems. The orthogonal 2D SPP only allows the placement of objects with sides parallel to the sides of the strip. The subclass of this algorithm is OF, according to the classification in [57] which deals with problems that require a fixed orientation of items (O) and no guillotine cut constraint (F).

The algorithm has parallels to the IBF algorithm with respect to the feature of performing a two-stage intelligent search in which a scoring rule is utilised to identify the best item to pack in the available space. The algorithm evaluates each item to pack,  $i$ , and assigns it a score from 0 to 4 depending on the given available packing space  $s$ . The item with the highest score for the current available space will be packed. The algorithm packs items into the lowest leftmost space possible and adopts the notion of a skyline by Burke *et al.* to define all available spaces. If no unpacked item fits into the lowest leftmost packing space  $s$ , then the items are re-evaluated for the next available lowest and leftmost space. The number and size of available spaces are then updated and the procedure is repeated until all items are packed in the strip.

Borrowing from the notation introduced earlier in the description of the IBF algorithm, each available space  $s$  is denoted by its leftmost point  $x$ , its lowest point  $y$ , the width  $\omega$

of the space, the height of its left wall  $h_1$  and the height of its right wall  $h_2$ . Each of the  $m$  available packing spaces are saved in an array  $S$  in increasing order of  $x$ -coordinate. For a given item  $i$  and space  $s$ , the scoring rules are computed according to two cases and are determined based on the rules in Table 2.2. The length and width of item  $i$  are given as  $length[i]$  and  $width[i]$  respectively and fitness  $f$  represents the score of the unpacked item  $i$  for the available space  $s$ .

Table 2.2: Scoring rules for evaluating items and calculating change in number of available spaces  $s$  in the CH algorithm.

Case	Condition	$f$	$m$
$h_1 \geq h_2$	$\omega = width[i]$ and $h_1 = length[i]$	4	-1 or -2
	$\omega = width[i]$ and $h_1 < length[i]$	3	0
	$\omega = width[i]$ and $h_1 > length[i]$	2	0
	$\omega > width[i]$ and $h_1 = length[i]$	1	0
	$\omega > width[i]$ and $h_1 \neq length[i]$	0	+1
$h_1 < h_2$	$\omega = width[i]$ and $h_2 = length[i]$	4	-1
	$\omega = width[i]$ and $h_2 < length[i]$	3	0
	$\omega = width[i]$ and $h_2 > length[i]$	2	0
	$\omega > width[i]$ and $h_2 = length[i]$	1	0
	$\omega > width[i]$ and $h_2 \neq length[i]$	0	+1

The chosen item is packed into the space on the side with the tallest wall (side). For the case where  $h_1 \geq h_2$  (i.e., the left-hand wall is greater than the right-hand wall) items are packed on the left-hand side of the available space. When the right-hand wall of the space  $s$  is greater than the left-hand wall ( $h_1 \leq h_2$ ), the items are packed on the right-hand side of the space. Depending on the score of the item that is packed, the number of available packing spaces  $m$  in the space array  $S$  changes by +1, -1, -2 or it remains unchanged. For illustration of the scoring rule, refer to Figure 2.9.

Figure 2.9(a) shows the structure of the space  $s$ . An item with dimensions  $h_1$  and  $\omega$  will have a score of 4 for space  $s$  as shown in Figure 2.9(b) and the number of available spaces  $m$  will decrease by 1. Figure 2.9(c) shows an item with  $width[i] = \omega$  and  $length[i] > h_1$  which will have a score of 3. Figure 2.9(d) and 2.9(e) shows that an item  $i$  with  $width[i] = \omega$  and  $length[i] < h_1$  will have a score of 2 and an item with  $length[i] = h_1$  and  $width[i] < \omega$  will have a score of 1 respectively. Neither item packed into space  $s$  will change the number of available spaces  $m$ . An item with  $width[i] < \omega$  and  $length[i] \neq h_1$  will have a score of 0 as shown in 2.9(f) and will increase  $m$  by 1.

The algorithm is given in pseudocode listing form in Algorithm 3.

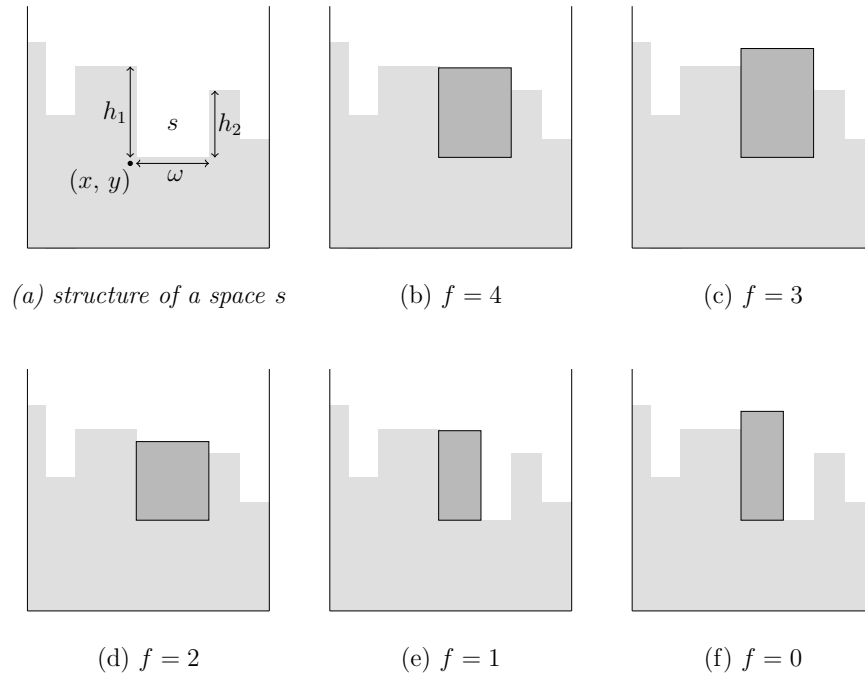


Figure 2.9: Example of the scoring rule in the CH algorithm for the case when  $h_1 \geq h_2$  (Leung *et al.* [53]).

Application of the CH algorithm outlined above is used for packing the items in the set  $\mathcal{H}$  for different orderings of the input sequence of the rectangles.

**Example 2.4.** The items in the set  $\mathcal{H}$  of Table 2.1 are left in the default order,  $\mathcal{H} = \{\mathcal{H}_1, \dots, \mathcal{H}_{10}\}$ . Initially, the lowest available space is simply the floor of the strip. Item  $\mathcal{H}_2$  is the first to be packed in accordance with the scoring rule;  $\mathcal{H}_2$  is the item with score 2 while all other items have a score equal to 0 with respect to that space. It has a width matching exactly the floor of the space and the placement is easy. The skyline is updated so that the only available space is the area above  $\mathcal{H}_2$ . Searching through the unpacked item list and assessing fitness, none of the items have a non-zero score. Since there is a tie amongst these items, the first item in the ordered is selected to be packed and in this case corresponds to  $\mathcal{H}_1$ . This item is placed left-justified in the corner. This placement results in two new available spaces: the area above  $\mathcal{H}_1$  and the top edge of  $\mathcal{H}_2$  not sitting beneath  $\mathcal{H}_1$ . The latter is the lowest space and, again, none of the remaining unpacked items have a non-zero score. Item  $\mathcal{H}_3$  being now the first in the list is selected as the best item that fits the space. The CH algorithm chooses to pack beside the taller neighbour and so  $\mathcal{H}_3$  is packed right-justified in the space. This continues similarly to pack items  $\mathcal{H}_8$  and  $\mathcal{H}_4$  next; the scores for these items in the steps being 1 and 3 respectively.

At this point, the number of available spaces is three. The region above  $\mathcal{H}_1$  is the lowest and in the steps that follow items  $\mathcal{H}_5$ ,  $\mathcal{H}_7$  and  $\mathcal{H}_{10}$  are stacked on top of each other above



$\mathcal{H}_1$  with each of these items being scored as 2.

The number of available spaces is now reduced to two. The region above  $\mathcal{H}_8$  and  $\mathcal{H}_3$  is the lowest with only the items  $\mathcal{H}_6$  and  $\mathcal{H}_9$  still to pack. Both items have a score of 0 and since  $\mathcal{H}_6$  is ordered in the position before  $\mathcal{H}_9$  in the list it is selected for placement. The tallest neighbour of this space is the boundary of the strip and hence  $\mathcal{H}_6$  is placed right-justified in the corner of the space. The lowest available space becomes the area sitting left of  $\mathcal{H}_6$  and the width of  $\mathcal{H}_9$  matches exactly the width of the space and hence  $\mathcal{H}_9$  is placed here. All the items have been packed.

The final packing layout is shown in Figure 2.10. This packing arrangement results in a height of 40 which is a match for the known optimal height of 40.

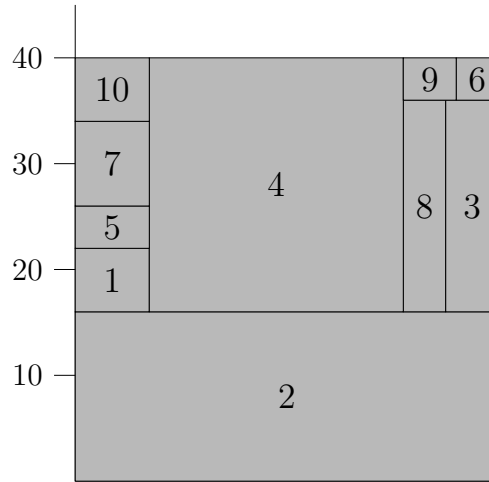


Figure 2.10: Result of packing the items  $\mathcal{H}$  using the CH algorithm. The items are presented to the algorithm in the default order. The resulting packing height is 40.

**Example 2.5.** The items in the set  $\mathcal{H}$  of Table 2.1 are sorted according to decreasing height and resolving ties by additionally sorting these items according to the default ordering yields the list  $\mathcal{H}' = \{\mathcal{H}_4, \mathcal{H}_3, \mathcal{H}_8, \mathcal{H}_2, \mathcal{H}_7, \mathcal{H}_1, \mathcal{H}_{10}, \mathcal{H}_5, \mathcal{H}_6, \mathcal{H}_9\}$ . The packing of the set  $\mathcal{H}'$  is performed similarly as in the example previous with the CH algorithm. The order of the packing of the rectangles is  $\mathcal{H}_2, \mathcal{H}_4, \mathcal{H}_3, \mathcal{H}_8, \mathcal{H}_7, \mathcal{H}_1, \mathcal{H}_{10}, \mathcal{H}_5, \mathcal{H}_6, \mathcal{H}_9$ . The final packing layout is shown in Figure 2.11. This packing arrangement coincides with the known optimal height of 40.

**Example 2.6.** The items in the set  $\mathcal{H}$  of Table 2.1 are sorted according to decreasing width and resolving ties by additionally sorting these items according to the default ordering yields the list  $\mathcal{H}^* = \{\mathcal{H}_2, \mathcal{H}_4, \mathcal{H}_1, \mathcal{H}_5, \mathcal{H}_7, \mathcal{H}_{10}, \mathcal{H}_3, \mathcal{H}_9, \mathcal{H}_6, \mathcal{H}_8\}$ . The packing of the set  $\mathcal{H}^*$  is performed similarly as in the example previous with the CH algorithm. The order of the packing of the rectangles is  $\mathcal{H}_2, \mathcal{H}_4, \mathcal{H}_1, \mathcal{H}_5, \mathcal{H}_3, \mathcal{H}_8, \mathcal{H}_7, \mathcal{H}_{10}, \mathcal{H}_9, \mathcal{H}_6$ .

The final packing layout is shown in Figure 2.12. The produced packing layout gives rise to a packing height of 44, slightly above the optimal height 40.

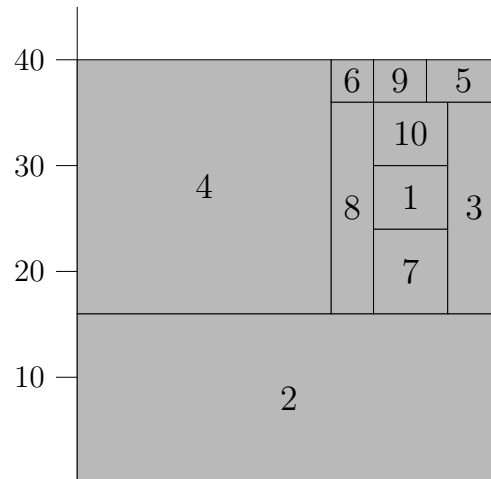


Figure 2.11: Result of packing the items  $\mathcal{H}$  using the CH algorithm. The items are presented to the algorithm by sorting the items in  $\mathcal{H}$  by decreasing height. The resulting packing height is 40.

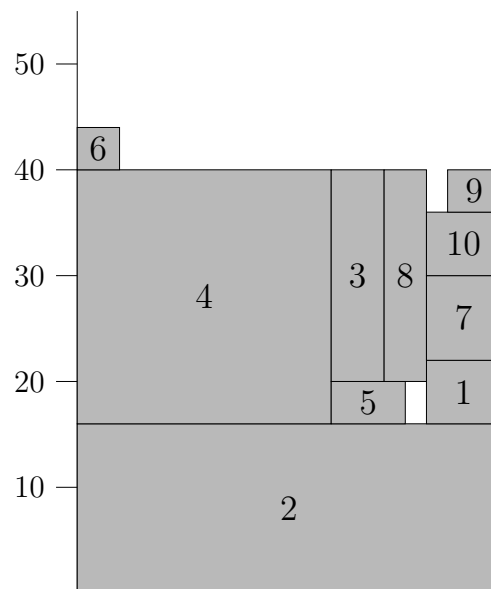


Figure 2.12: Result of packing the items  $\mathcal{H}$  using the CH algorithm. The items are presented to the algorithm by sorting the items in  $\mathcal{H}$  by decreasing width. The resulting packing height is 44.

---

**Algorithm 3:** Constructive Heuristic

---

**Input** : A list of  $n$  items to be packed,  $\mathcal{R}$ ; the dimensions of the items,  $\langle width[i], length[i] \rangle$ ; and the strip width  $W$ .

**Output** : A feasible packing arrangement of the items into a strip of width  $W$ , and the height of the packed objects  $h$ .

```

1   $h \leftarrow 0, packedItems \leftarrow 0$ ;
2  initialise the space array  $S$ ;
3  while  $packedItems < n$  do
4      find the lowest and leftmost space  $s = (x, y, \omega, h_1, h_2)$ ;
5      if an item in  $\mathcal{R}$  fits into  $s$  then
6          for each item  $i$  in  $\mathcal{R}$  do
7               $s_i = ConstructiveScore(i, \omega, h_1, h_2)$ 
8          end
9          Select an item  $r$  such that  $s_r = \max_i \{s_i\}$ ;
10         if  $length[r] + y > h$  then
11              $h \leftarrow length[r] + y$ 
12         end
13         if  $h_1 \geq h_2$  then
14             pack item  $r$  against left wall and remove  $r$  from  $\mathcal{R}$ ;
15              $packedItems \leftarrow packedItems + 1$ ;
16             update space array  $S$ ;
17         else
18             pack item  $r$  against right wall and remove  $r$  from  $\mathcal{R}$ ;
19              $packedItems \leftarrow packedItems + 1$ ;
20             update space array  $S$ ;
21         end
22     else
23         update space array  $S$ ;
24     end
25 end
26 return  $h$ ;

```

---

## 2.3 Chapter Summary

In this chapter, a general overview of the heuristics from the literature for solving packing problems (approximately) dating as far back as the 1980s to more recent times were described. Thereafter, the scope of this chapter was delimited to introduce and review in detail the BF algorithm by Burke *et al.* [15] in Section 2.1.1, an updated and improved BF algorithm by Wei *et al.* [93] in Section 2.1.2 and finally the CH algorithm by Leung *et al.* [53] in Section 2.2.

The problem instance of Burke *et al.* [15] in Table 2.1 was introduced to illustrate the working of the algorithms presented in this chapter. It was found that for this problem instance with known optimal height of 40, presenting the IBF and CH algorithms the default order of the items as specified by the problem instance yielded resultant packing heights of 60 and 40 respectively – the packing layouts are provided accordingly in Figure 2.6 and Figure 2.10. However, sorting the instance according to decreasing height initially and thereafter presenting this sorted list to both algorithms yields the known optimal height of 40 for both, given by the packing arrangement in Figure 2.7 and Figure 2.11. Similarly, sorting by decreasing width yielded packing heights of 46 and 44 respectively for the IBF and CH algorithms, given in Figure 2.8 and Figure 2.12. These results are summarised in the table below, Table 2.3.

Table 2.3: The heights of the packing layout of the items in  $\mathcal{H}$  generated by the heuristic algorithms for different initialised packing orders.

Heuristic	Default	Decreasing height	Decreasing width
Improved Best-Fit	60	40	46
Constructive	40	40	44

---

## CHAPTER 3

---

### Metaheuristic Algorithms

In the previous section, in the packing of items in  $\mathcal{H}$  the optimal packing height is achieved using the IBF algorithm through a decreasing height initial sorting order, and as well as the CH algorithm via both the default and a decreasing height sorting order. Although the optimal packing is achieved in these cases, the packing layout for each of these differs. An important take-away from this is that the optimal packing arrangement for a particular problem instance may not necessarily be unique. Moreover, the initial sequence order of the items to pack when presented to each of the algorithms does have significant impact on the final packing arrangement. It was demonstrated that for different sortings of the input sequence of rectangles the heuristic algorithms in the chapter previous yielded differing packed solutions. The naturally arising question at this stage is to identify an optimal or best ordering of the input sequence of the rectangles for packing using the heuristics, if it exists. A population-based search or a local search can be used to improve the solution.

It is quite straightforward to draw the comparison of searching for an optimal solution to that of treasure hunting. Suppose that we wish to hunt hidden treasure in a hilly landscape within a fixed interval of time. It is possible to conceive a scenario where such a search is conducted blindfolded and devoid of any guidance, a pure random search. With the right amount of luck it is possible to find the treasure, however this is highly unlikely and not an efficient process. In stark contrast, if we have information that the treasure has been placed at the highest peak of a known region, then certainly the strategy would be of directly ascending the slope and attempting to reach the summit. Such a scenario is in alignment with the classic hill-climbing techniques. Generally speaking, the problem is a mixture of these two extremities, that is to say we are neither blindfolded and neither do we have perfect information as to where to look.

Naturally, a brute-force strategy whereby every square metre is searched will always theoretically guarantee success in a well-defined search space, but such an approach should be saved as a last resort owing to time constraints. A more rational strategy would be to refine the search strategy and search space. This often begins with a random walk while piecing together clues and hints, and building on previous knowledge. We inspect possi-

ble spaces initially at random before proceeding to a more plausible location. If we wish to hunt alone, the entire path traversed is a trajectory-based search. We may, instead, also gather individuals together to perform the search and to aggregate information, a population-based search. These individuals differ in that some perform better than others; naturally, we wish to replace those who under-perform (perhaps after an evaluation of their fitness) with new individuals. The desire at every step of this search is to find the treasure in the shortest amount of time. A random walk procedure defined as in the cases here constitutes the backbone of many modern metaheuristic search algorithms.

Any metaheuristic algorithm consists of two key features, namely, *exploitation* and *exploration*. Exploration refers to the need to randomly generate diverse solutions so as to step sufficiently through the search space on a macro level, while exploitation is the micro-level process of honing in on a particular local region in search of better solutions. In conjunction, both these are critical components in the procedure; exploration prevents being trapped at local optima and exploitation ensures convergence to optimality [99]. An important remark is that an algorithm achieves good performance if these components are well balanced. If exploration is done too freely whilst there is too low exploitation then the search path is too wide and convergence is slow. In contrast, placing too much importance on exploitation over exploration yields quicker convergence, but bears the consequence of decreasing the probability of finding the true global minimum.

The steps that frame these algorithms and their adaptations are usually given as follows:

- 1) propose an initial solution;
- 2) propose a candidate solution;
- 3) evaluate the candidate solution against the current solution;
- 4) based on the evaluation, either reject the candidate solution or replace the current solution with the candidate; and
- 5) repeat steps 1-3 until a stopping criterion is satisfied.

A commonly made distinction of metaheuristic algorithms is by classification of these methods into distinct classes, namely *population-based* or *trajectory-based methods*. The former concerns a set of solutions (the population) at every step, and the latter describes a search process that is characterised by a direction (trajectory) in the search space.

Deviating from the abstraction, these algorithms are well-suited for packing problems. In this context, 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 as decoded by a heuristic routine, for instance the CH algorithm from [Chapter 2](#) [74].

Genetic algorithms (populated-based) and simulated annealing (trajectory-based) are the most utilised metaheuristic algorithms for packing problems [16, 17, 43, 50]. We will briefly explore these as generic methods before further detailing their respective implementation for the 2D SPP. For each algorithm, the aforementioned concepts of exploration and exploitation are tuned by a set of hyperparameters. Various studies have been conducted that show that the choice of parameter values in the metaheuristic techniques employed in hybrid SPP algorithms has notable impact on their performances [43, 53]. While the aim of this project is not to optimise for these parameters for any given problem instance (this is a difficult challenge in and of itself and remains, in fact, still an open problem in the literature), we will provide insight as to how to select appropriate parameters and to summarise some of the findings found in the literature.

For illustrative purposes, the problem instance of Bengtsson in his set of ten benchmark instances (often referred to as *beng*) proposed in 1982 [8] will be referred to in generating example packing solutions using the algorithms of this chapter. Specifically, the first of these ten instances is considered (*beng1*) and the items contained in this problem instance will be referred to by the set given as  $\mathcal{I}$ . Bengtsson details the construction of each of the rectangles  $i$  in all ten of these instances by the assignments

$$height_i := \lceil 12\gamma_{h,i} + 1 \rceil \text{ and } width_i := \lceil 8\gamma_{w,i} + 1 \rceil$$

where the values  $\gamma_h$  and  $\gamma_w$  are random uniform numbers from the interval  $(0, 1)$  and  $\lceil \cdot \rceil$  denotes the function  $\lceil \cdot \rceil : \mathbb{R} \rightarrow \mathbb{Z}$  which maps a real number to its nearest integer. For this particular problem instance, there are 20 items to be packed into a strip of width 25 with the optimal packing arrangement known to produce a height of 30. Bengtsson's instances allow for 90 degree rotations, however this chapter limits the scope to only fixed orientations of the items. This instance, moreover, the set of instances, was obtained from [89]. The dimensions of each of the items contained in the set  $\mathcal{I}$  are presented in Table 3.1, and an illustration of these items in Figure 3.1. A different problem instance, arguably a more challenging one considering the number of items to pack in  $\mathcal{I}$  is twice that of  $\mathcal{H}$ , is preferred over the earlier considered item set  $\mathcal{H}$  of Burke *et al.* considering the known optimal height was achieved quite simply by initially sorting the order of items by decreasing height and then applying both heuristic algorithms accordingly. It is noted that these instances of Bengtsson have been used by numerous authors in the study of the relative performance of algorithms designed for the SPP [2, 53, 95, 98].

Table 3.1: Dimensions  $(h(\mathcal{I}_i), w(\mathcal{I}_i))$  of rectangles  $\mathcal{I}_1, \dots, \mathcal{I}_{20}$  used as an example instance.

Items, $\mathcal{I}_i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Height, $h(\mathcal{I}_i)$	8	11	10	4	12	11	12	3	4	3	9	1	8	11	11	9	9	11	1	7
Width, $w(\mathcal{I}_i)$	6	4	3	7	6	6	8	6	3	8	1	3	6	3	1	8	1	5	7	8

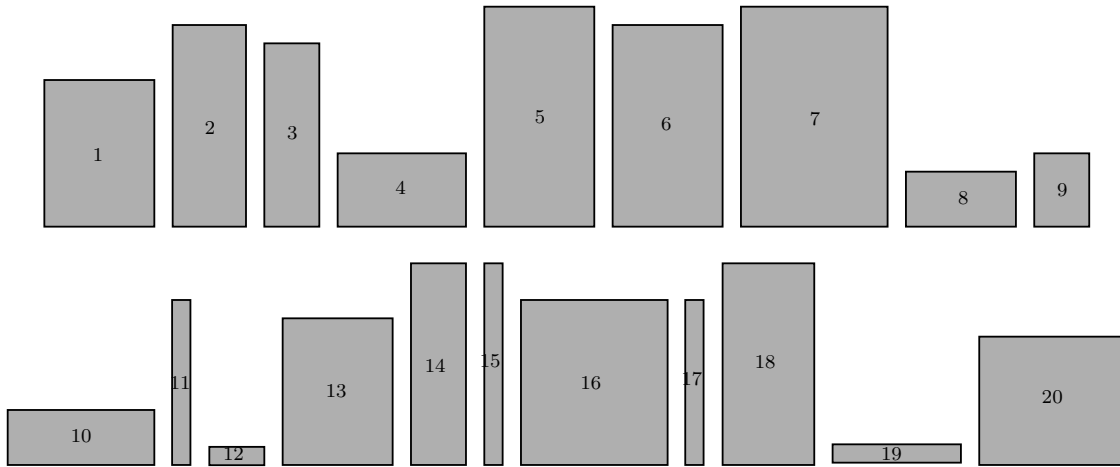


Figure 3.1: A scaled graphical representation of the items in  $\mathcal{I}$ .

### 3.1 Hybrid Genetic Algorithms

Genetic algorithms (GAs) were first used for the 2D SPP in 1985 [24, 87]. Hopper and Turton [44] provide an extensive overview of GA implementations that have been developed for the 2D SPP and its variants. The authors categorise these implementations into three classes.

1. The first is the hybrid approaches where a GA is used to search for the packing order of items and a heuristic algorithm is used as a placement routine to decode the packing list into a 2D SPP solution.
2. The second category contains hybrid approaches that try to capture additional layout information in the encoding technique.
3. The third class operates without any encoding of solutions, instead solving the problem directly in the 2D space. This approach works by taking a starting solution and relocating items within the layout accepting moves that increase the fitness value or keeps it the same.

We will be focusing on the first category, hybrid genetic algorithms.

GAs are population-based algorithms. They start with an initial population of feasible solutions known as *chromosomes*. The optimisation process, based on *natural selection*, involves the evolution of the initial population solutions through time using the genetics-inspired operators of crossover and mutation. The selection operator selects individuals in the population who will reproduce based on a given selection method. In general, fitter individuals have a higher chance of being chosen to reproduce and hence produce more offspring than less fit individuals. Crossover allows for the recombination or exchange of parts between chromosomes, while mutation randomly changes some of



the *gene* values (a single value in the chromosome) in the chromosome [64].

Hybrid GAs are a common class of algorithms used for the 2D SPP. [44] In the context of strip packing, the solution is often represented by a packing permutation. The permutation represents the encoding of solutions and it is the initial order of items to be packed. [74] Hybrid GAs involve a two-step process where the GA first exploits the encoded solutions. Using a decoding algorithm (heuristic algorithm), the solutions are transformed into physical packing layouts which are then evaluated for solution quality based on some fitness function. A common measure used to determine the solution quality of a heuristic packing layout is the packing height of the resulting layout. Thus, the choice of the heuristic packing algorithm plays a large role in the effectiveness of hybrid GAs. One benefit of using a decoding algorithm is that it allows for domain specific knowledge to be incorporated into the procedure which in turn reduces the size of the search space as only viable solutions will be provided by the decoding algorithm. The search space of a 2D SPP is the number of possible packing arrangements of the items. Hybrid GAs explore the search space to find the best possible packing order to place items within the strip. One downside to using a decoding algorithm in hybrid GAs is the loss of information about the layout of the data structures (*genotype*) that the GA operates upon. Therefore, some of the information used in the decoding of the solution (*phenotype*) is not available to the genetic operators which means that this information cannot be passed on to subsequent generations. [44]

Rakotonirainy [74] gives a detailed description of the most commonly used variables in GAs for use in the 2D SPPs. The decisions to make regarding the design of the hybrid GA include the chosen solution representation technique, the size and method of generating the initial solution population, the fitness measure to use, the decoding algorithm, genetic operators, and a termination criterion.

The initial population consisting of a set of starting solutions can be generated randomly or can be seeded using a packing rule to generate better quality starting solutions to help the GA. The population size generally stays fixed throughout the process.

The success of GAs depends largely on their ability to balance selective pressure with maintaining population diversity [13]. A good population will retain historical information while also having enough diversity for the GA to explore new solution spaces. During the selection for the reproduction phase, parent solutions are selected to mate based on their fitness. Parent selection based on fitness allows the population to retain historical information as good parents are more likely to mate and pass on their genetic information to their offspring which in turn are more likely to be selected as they will more often than not resemble their parents. Selection procedures should not allow for individuals with poor fitness to be overselected as this will decrease information retention. On the other hand, overselection of good quality individuals will lead to a reduction in population variance.

A common selection procedure previously used in SPPs is *proportional selection*, applied by Hopper and Turton [44] and Jakobs [46]. The probability that a chromosome is selected for reproduction is proportional to its fitness. The fitter an individual is, the more likely they are to be chosen for reproduction. *Roulette wheel selection* is a common way of implementing proportional selection. Each individual  $i$  is assigned to an area  $A_i$  of the circle, much like a roulette wheel, where the area is proportional to the fitness value of the individual. The probability of selection for reproduction for the  $i^{th}$  individual is given as

$$p_i = \frac{f_i}{\sum_{j=1}^N f_j} > 0 \quad (3.1)$$

where  $p_i$  is the probability of selection and  $f_i$  is the fitness in a population of size  $N$ . The interval  $I = [0, 1)$  is divided into  $N$  subintervals and each individual is assigned to a subinterval

$$\begin{aligned} A_1 &\leftrightarrow I_1 = [0, p_1), \\ A_2 &\leftrightarrow I_2 = [p_1, p_1 + p_2), \\ &\vdots \\ A_N &\leftrightarrow I_N = [1 - p_N, 1). \end{aligned}$$

Random numbers are drawn uniformly in the interval  $[0, 1)$  and the probability intervals determine which individuals are selected for reproduction.

A drawback of proportional selection is that if a population is dominated by one or a few individuals with disproportionately large fitness values, proportional selection may overemphasise the selection of the fittest individuals as other individuals will have very low selection probabilities. This may lead to a lack of genetic diversity in the population [74].

Another common GA selection method is *tournament selection*. Basic tournament selection works by randomly selecting  $k$  individuals with replacement from the current population  $N$  ( $k < N$ ) for consideration as parent chromosomes. The individual with the best fitness out of the  $k$  individuals in the tournament is selected to reproduce [31].

Crossover and mutation are then applied to the selected candidates with some probability. Crossover operators that have been used in SPP literature are 1-point Order Crossover (OX) [46, 87], 2-point Order Crossover [56], and Partially Matched Crossover (PMX) [44, 54].

*Order Crossover*<sup>1</sup> involves splitting the selected parents into three components based on two randomly selected points [100]. The middle component is matched between each parent and mapped to its offspring. This means that the middle section from parent 1 will be mapped to offspring 2, likewise for parent 2 and offspring 1. The rest of the chromosome is filled based on the relative order of genes in the parent chromosomes; offspring 1 with parent 1 and similarly offspring 2 with parent 2. The relative ordering rearranges the parent chromosome so that it starts from the first split point. An example of order crossover is shown in Figure 3.2.

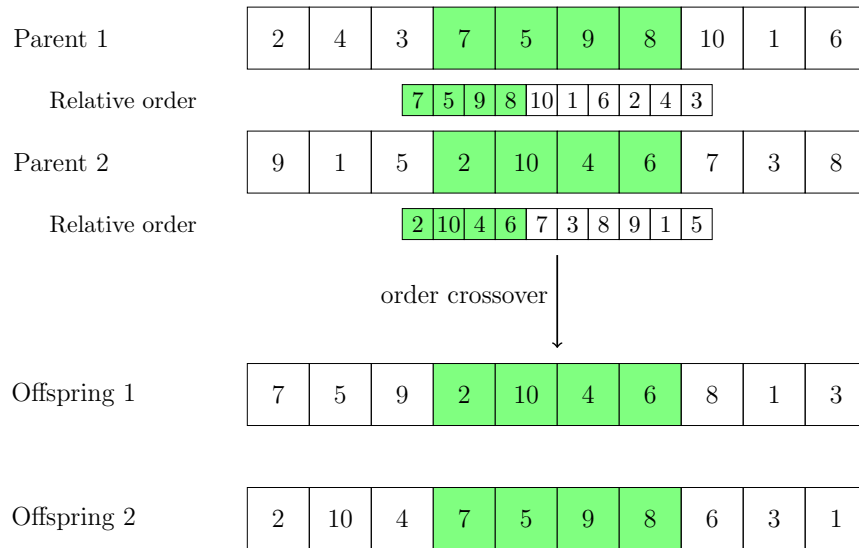


Figure 3.2: Example of Order Crossover. The first split point occurs after the 3<sup>rd</sup> gene in the parent chromosomes.

*Partially Matched Crossover* employs the use of two randomly chosen cut points along the encoding of the parent solutions. The section between the two cut points defines a matching section for which each parent maps to the other. For the example given in Figure 3.3, the mapping is  $7 \leftrightarrow 2$ ,  $5 \leftrightarrow 10$ ,  $9 \leftrightarrow 4$  and  $8 \leftrightarrow 6$ . The matching sections are swapped between offspring and the remaining sections are filled from their corresponding parent. If a duplicate number occurs in the offspring chromosome, the duplicate is replaced based on the mapping defined for the matching section.

Once the crossover operation is completed the mutation operator is applied with some probability. The mutation operator consists the change of part of a chromosome to generate new genetic characteristics and this can occur at a single site or multiple sites simultaneously. There are many methods for applying mutation in the literature, but *swap mutation* is commonly used in hybrid GAs for SPPs [44, 54]. This operator selects

<sup>1</sup>2-point Order Crossover is described.

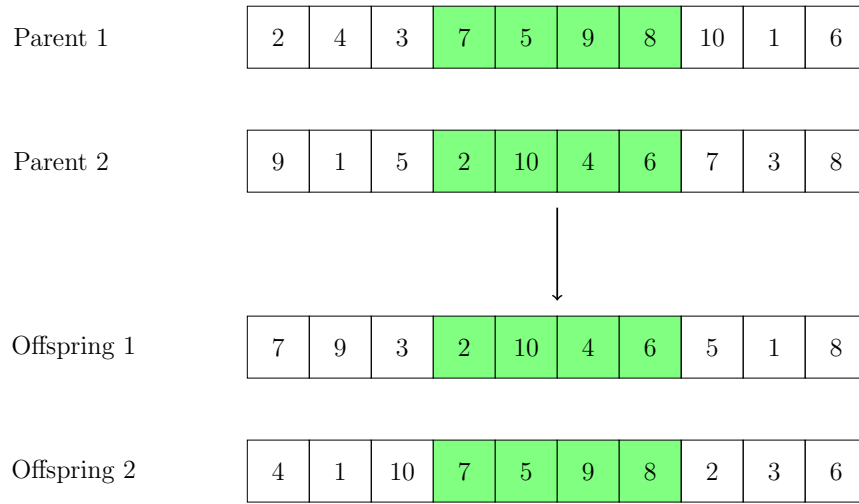


Figure 3.3: Example of Partially Matched Crossover.

two random genes in a solution encoding and their positions swapped to produce a new solution encoding. An illustration of such a swap is provided in Figure 3.4; here the first and fifth components have been selected on the left and the values exchanged to produce the solution encoding on the right.

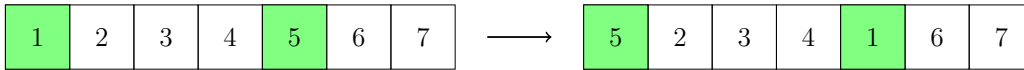


Figure 3.4: Example of swap mutation.

The principles of crossover and mutation are necessary components of the algorithm and address the notions of exploitation and exploration respectively in the context of converging towards the global optimum. Numerous investigations have been performed to determine ideal probability levels for crossover and mutation; the results of these suggest a relatively higher probability for crossover approximately in the range (0.6, 0.95), whereas the mutation probability is typically much lower, in the range (0.001, 0.05) [99].

Once crossover and mutation are completed, the new offspring's fitness is evaluated by the placement algorithm. Selection for replacement is the process that determines which individuals will survive into the next generation. The two main replacements techniques are *generational replacement* and *steady state replacement*. Generational replacement replaces the entire population at every generation while steady state replacement only replaces one (or a few) individuals in the current generation's population [74]. One method of steady state replacement is *elitist recombination* in which the two offspring compete with the parents to determine which individuals will be in the next generation. The best two out of the four individuals (two parents and two offspring) are included in the next generation [59]. Subsequent generations are produced similarly in an iterative

manner until some stopping criterion is met.

Typically, the termination criteria are based on convergence, a specification of the maximum number of iterations (generations) for the entire search process, or a budgeting of computational time.

A generic hybrid genetic algorithm implementation useful for the 2D SPP and as implemented in the decision support system is shown in [Algorithm 4](#).

---

**Algorithm 4:** Hybrid Genetic Algorithm

---

**Input** : A list of  $n$  items to be packed,  $\mathcal{R}$ ; the dimensions of the items,  $\langle width[i], length[i] \rangle$ ; and the strip width  $W$ .

**Output** : A feasible packing arrangement of the items into a strip of width  $W$ , and the height of the packed objects  $h$ .

- 1 generate  $N$  random permutations of items to be packed  $P_0$  (can be sorted according to some criteria);
  - 2 evaluate the fitness of each permutation (using a heuristic packing method) and store it;
  - 3 **while** *stopping criteria not yet met* **do**
  - 4     create a reproduction pool from  $P_t$  based on fitness and a chosen selection process;
  - 5     apply crossover and mutation to the reproduction pool (with probabilities  $p_c$  and  $p_m$  respectively);
  - 6     generate next population  $P_{t+1}$  of size  $N$  from population  $P_t$  and the reproduction pool using replacement strategy;
  - 7     save the best permutation found so far;
  - 8 **end**
- 

**Example 3.1.** *An example of packing solutions obtained by packing the items of instance  $\mathcal{I}$  of Table 3.1 is given in Figure 3.5. Two cases were considered: (a) packing of the items using the IBF heuristic by first ordering the items by decreasing height, and (b) using a hybrid GA to determine packing order and paired with the IBF algorithm decoding routine. In the latter, the mechanisms of roulette wheel selection, OX crossover, and elitism replacement operators were incorporated. A population size of 10 was specified, and the crossover and mutation probabilities were taken as 0.9 and 0.1 respectively. The fitness function of packing height associated with a solution was used in this implementation. The initialisation of the population was performed randomly and the algorithm was allowed to iterate over 50 generations and one of the best solutions generated is presented in Figure 3.5(b).*

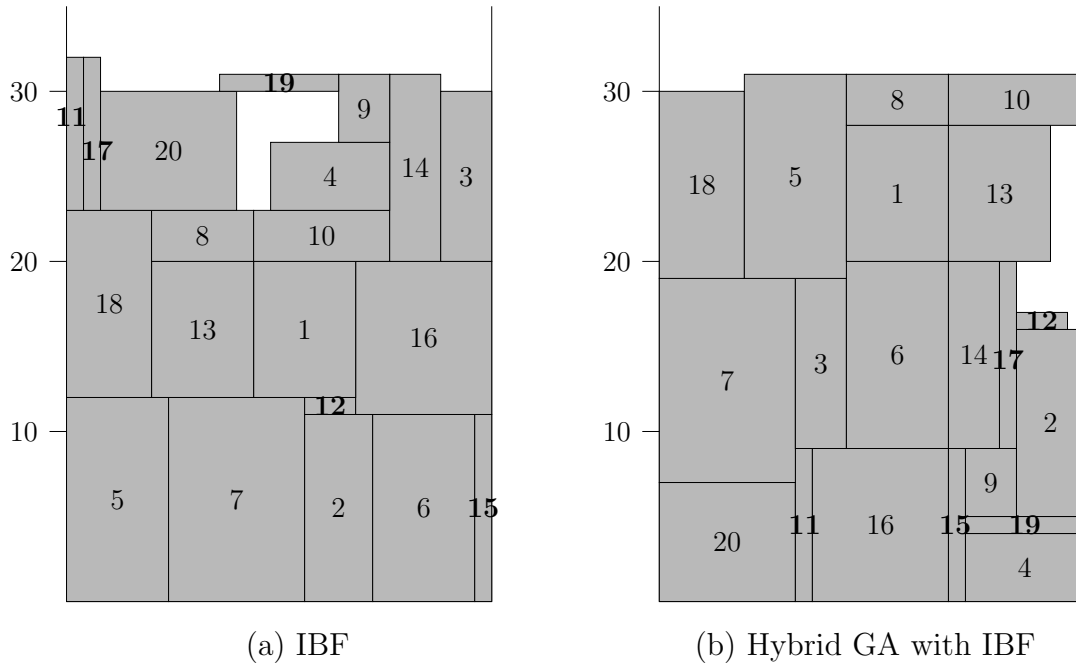


Figure 3.5: An example of a packing solution obtained with applying the IBF algorithm to the list of items  $\mathcal{I}$  sorted by decreasing height is given by (a), while an example of a packing solution obtained when applying the hybrid GA combined with the IBF algorithm as decoding routine to the list  $\mathcal{I}$  is given in (b). The resulting packing heights are 32 and 31 respectively.

## 3.2 Hybrid Simulated Annealing

In crystallography, a branch in the physical sciences concerned with the study of the arrangement and bonding of atoms to form crystalline structures and lattices, to grow a crystal, one starts by heating the raw materials to a molten state. In a liquid state the atoms have high energy and are randomly arranged, while a solid state corresponds to a minimum energy configuration of atoms. Once heated and in a liquid state the system is gradually cooled from the melting point until the crystal structure is frozen in. Throughout this procedure, a perfectly structured crystal (a minimum energy configuration) is formed as a result of slow and steady cooling. Too fast a cooling results in a structure that is frozen at a locally optimal metastable configuration of atoms, such as glass material or crystals with defects [29]. This described procedure, termed *annealing*, mimics various methods underpinning optimisation problems. In view of this connection, the various permitted states of the physical system correspond to the different feasible solutions of the combinatorial optimisation problem, and the energy of the system at any given particular state corresponds to the objective function value. It is based on this analogy of the physical annealing of solids that the concept of *simulated annealing*

was introduced in order to solve combinatorial optimisation problems.

Introduced first by Kirkpatrick in the early 1980s this method has since been a popular method in the class of trajectory-based metaheuristics [49]. This approach has seen wide application in problems ranging from the travelling salesman problem to solving Sudoku puzzles. The thread of commonality in these problems is that of combinatorial optimisation, and the consequence of which, to no surprise, implies that this technique can readily be extended to considering the problem of strip packing.

In broad terms, the simulated annealing (SA) algorithm typically commences with an initialisation of the starting parameters – a random initial solution and a relatively large starting temperature. At each iteration, a neighbour of the current solution is randomly generated by some suitably defined mechanism. The objective function is evaluated at this new point and thereafter the decision to accept or reject this as the new solution is performed probabilistically. Whilst running through the SA algorithm iteratively the temperature is cooled.

The overarching idea of the SA algorithm is that of a random search by means of a Markov chain, and compared to gradient-based methods and deterministic search methods SA avoids entrapment at local minima (for a minimisation problem) by not only accepting changes that improve the objective function but also keeps some changes that are not ideal. At high temperatures the algorithm explores successors wildly and randomly by modifying the search space in such a manner that makes unexplored areas more desirable. At each step in the iteration each candidate is evaluated against the current point with respect to the objective function. An improved or equal objective function value is always accepted, while worse solutions are accepted only with probability given as a function of temperature. In the case of the latter, this incorporation of a stochastic criterion for the acceptance of worse quality solutions is done in order to prevent the premature entrapment at local optima. The acceptance criterion is based on the Metropolis algorithm for the simulation of physical systems subject to a heat source (Metropolis *et al.* [63]). Progressively the algorithm steers towards being more selective of making worse-off moves decreases with decreasing temperature. Stabilisation is achieved at low temperatures and it is at this point that the procedure is terminated. The acceptance-probability function which captures this property at each iteration  $k$  is given by

$$p_k = \exp\left[-\frac{\Delta E}{T_k}\right], \quad (3.2)$$

where  $\Delta E$  is the change in the objective function value and  $T_k$  the temperature. This above function is one which has been adopted from and is based on the Boltzmann distribution, a distribution widely used in the field of thermodynamics [88].

As in the metaphor earlier of growing crystals described, a vital consideration in

any implementation of the SA algorithm is a clear definition of the function that manages changes in the temperature at each step of the iteration. This function is appropriately referred to as the cooling schedule. No single rule for optimal cooling and initial temperature combinations has been defined for the general optimisation problem. Numerous cooling schedules have been proposed in the literature, some of which include:

- *linear cooling*:

$$T_k = T_0 - \alpha k, \quad \alpha = \frac{T_0 - T_f}{N}; \quad (3.3)$$

- *exponential multiplicative cooling*, also referred to as *geometric cooling* (Kirkpatrick *et al.* [49]):

$$T_k = T_0 \alpha^k, \quad 0.7 \leq \alpha \leq 0.99; \quad (3.4)$$

- *logarithmic multiplicative cooling* (Aarts and Korst [1]):

$$T_k = \frac{T_0}{1 + \alpha \log(1 + k)}, \quad \alpha > 1; \quad (3.5)$$

- *Lundy and Mees cooling* (Lundy and Mees [60]):

$$T_k = T_{k-1}(1 + \alpha T_{k-1})^{-1}, \quad \text{very small } \alpha; \quad (3.6)$$

where  $T_0$  is the initial temperature,  $T_k$  the temperature at iteration  $k$ ,  $T_f$  is a temperature lower bound,  $N$  the maximum number of iterations, and  $\alpha$  the cooling parameter, or cooling rate. It is noted that in the above cooling schedules, the defined constraints for  $\alpha$  are not strict constraints but rather are recommended best practices for selecting  $\alpha$  in the respective cooling schedule so as to achieve optimal results.

In each cooling schedule there is an explicit definition of, at least, three parameters: initial temperature  $T_0$ , the temperature decrease function with an embedded cooling rate parameter  $\alpha$  governing the manner of temperature degradation, and the number of state transitions  $L$ , also known as the length of epoch, to be performed for each temperature. These are determined empirically, and hence are tuning parameters and all of which have important and significant impact on the algorithm result.

The manner in which the acceptance-probability function is defined has enormous significance in terms of specifying the right initial temperature. For a given change  $\Delta E$ , if  $T$  is too large ( $T \rightarrow \infty$ ), then  $p \rightarrow 1$  which implies most changes will be accepted, i.e., exploitation is low. Conversely, for  $T$  too low ( $T \rightarrow 0$ ), then for any  $\Delta E > 0$  (a worse solution in the context of minimisation) then the candidate solution is unlikely to be accepted since  $p \rightarrow 0$ , and so diversity of solutions is poor i.e., exploration is low. The key to a good starting temperature is one which allows for inferior solutions with very high probability initially for a good length of time before decaying to zero thereafter.



Critical information in this decision-making process is captured by the objective function; the range of values  $\Delta E$  can take on hints at the appropriate magnitude of  $T_0$ .

$L$  is also a noteworthy parameter. For each temperature, multiple evaluations of the objective function are required. If there is an insufficient number of evaluations performed then there is the risk that the system will not stabilise and consequently convergence to the global optimal is difficult [99]. Naturally, the ideal scenario would be to have  $L$  be set to as large a value as possible, but this is more often than not computationally infeasible. There are two main ways to define the number of iterations at each temperature: constant or varied. The latter case specifically is defined in a manner with the intention of increased iterations at a lower temperature to better explore local minima.

Both the form of the mathematical function and the cooling parameter control the temperature decrease. In Figure 3.6 a plot of the Equations 3.3 - 3.6 is illustrated for  $T_0 = 100$  and  $\alpha = 0.9$ , while in Figure 3.7 the same equations plotted with  $T_0 = 100$  and  $\alpha$  values varied and selected according to the recommended values in the literature. Both are plotted for 50 temperature updates,  $N = 50$ . The selection of the function and cooling parameter are based on the desired behaviour for the temperature over time. It is common, however, in the SPP literature to use a geometric schedule [16, 43, 53]. For instance, as observed in the graphs, the logarithmic function decays rapidly compared to the geometric function initially but eventually converges to a temperature larger than the geometric function, while the linear function defines a steady and constant decrease in temperature. The Lundy and Mees function behaves similarly to the logarithmic function initially but thereafter decays to 0.

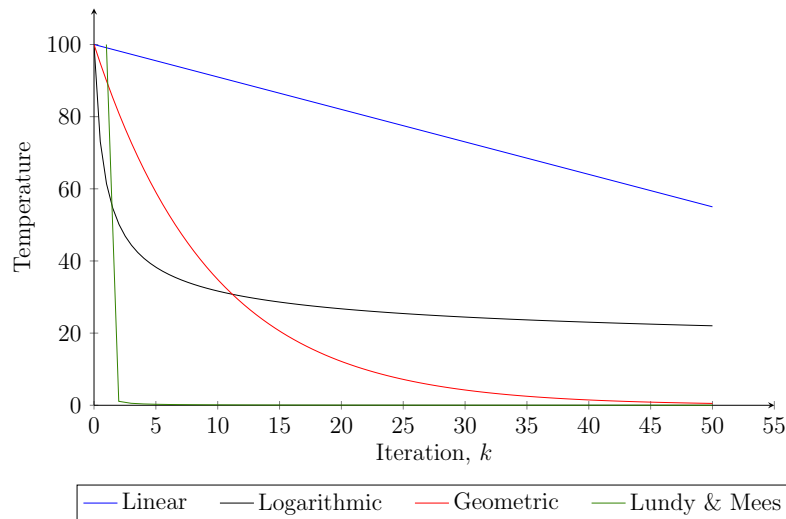


Figure 3.6: Temperature decline by the linear, logarithmic, geometric, and Lundy & Mees cooling schemes with parameters  $T_0 = 100$  and  $\alpha = 0.9$ .

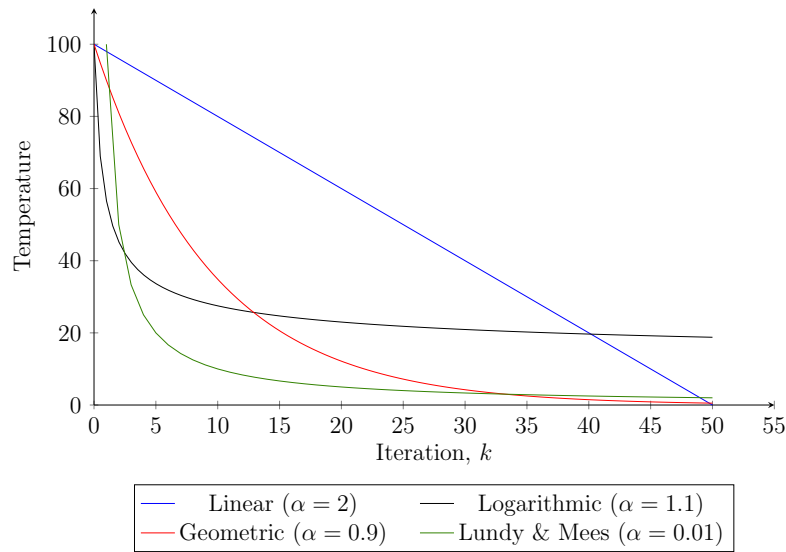


Figure 3.7: Temperature decline by the linear, logarithmic, geometric, and Lundy & Mees cooling schemes with parameters  $T_0 = 100$  and various  $\alpha$ .

With plenty of parallels to the hybrid GA for SPPs, 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 decoding routine. The SA algorithm attempts to determine a good ordering for which the items are to be packed, while the heuristic decoding routine serves to interpret the permutation and to evaluate the quality by transforming it into a packing layout [44].

The hybrid SA method represents a packing problem by a permutation that is interpreted as the order in which the rectangles are packed [43]. The algorithm begins the search by initialising at random a feasible packing solution (i.e., a packing permutation) and a selection of an initial temperature. Using a pre-specified heuristic routine, the quality of the packing permutation is assessed. A manipulation operator is applied to the solution to generate a new candidate solution. The manipulation operator consists of selecting at random a new candidate solution in the neighbourhood of the current. The neighbourhood of a packing solution encoding consists of the set of all possible candidate solution encodings from the current solution. The elements of this set are the permutations that result from randomly selecting a pair of items in the current permutation at random and reversing their order [16, 43, 53]. This is analogous to the order-based mutation operator encountered in the GA. Using the heuristic packing routine to evaluate quality, the new candidate solution is compared to the current solution. The criterion often employed is the comparison of packing heights. If the candidate solution yields an improved packing layout it becomes the new solution; however, if the new candidate solution results in a worsening solution it is then only accepted as the new solution if the probability condition, characterised by the cooling schedule, is satisfied.

At the current temperature the steps of applying the manipulation operator and deciding

on whether to accept or reject the candidate is repeated in accordance with the defined epoch length. The temperature is thereafter updated and the process is repeated at the new temperature. The procedure terminates upon satisfying a termination criterion. The criteria most often utilised involve specifying a maximum number of iterations that may be performed, specifying a lower bound on the temperature, or a criterion for convergence.

A pseudocode listing of the hybrid SA algorithm is given in [Algorithm 5](#).

---

**Algorithm 5:** Hybrid Simulated Annealing Algorithm

---

**Input** : A list of  $n$  items to be packed,  $\mathcal{R}$ ; the dimensions of the items,  $\langle width[i], length[i] \rangle$ ; and the strip width  $W$ .

**Output** : A feasible packing arrangement of the items into a strip of width  $W$ , and the height of the packed objects  $h$ .

```

1 generate an initial packing permutation and set as the current as well as the best solution;
2 evaluate the permutation fitness using a heuristic method;
3 generate an initial temperature and set as the current temperature;
4 loop until stopping criterion met
5   for the length of epoch do
6     determine neighbourhood structure of the current solution;
7     randomly select a neighbouring solution;
8     set the current to the neighbouring solution;
9     if current solution is better than best solution then
10      the best solution is the current;
11   else
12     if probability of accepting a worsening solution is satisfied then
13       the best solution is the current;
14     end
15   end
16 end
17 update the temperature;
18
19 end
```

---

**Example 3.2.** An example of packing solutions obtained by packing the items of instance  $\mathcal{I}$  of Table 3.1 is given in Figure 3.8. Two cases were considered: (a) packing of the items using the CH heuristic by first ordering the items by decreasing height, and (b) using a hybrid SA to determine packing order and paired with the CH algorithm decoding routine. In the latter, a logarithmic type cooling schedule was selected with cooling parameter 1.1. The initial temperature was taken as 20 and the number of state transitions at each temperature set to 30. The fitness function of packing height associated with a solution was used in this implementation. The initialisation of the starting permutation was performed randomly and the algorithm was allowed to perform a maximum of 100 iterations and one of the best solutions generated is presented in Figure 3.8(b).

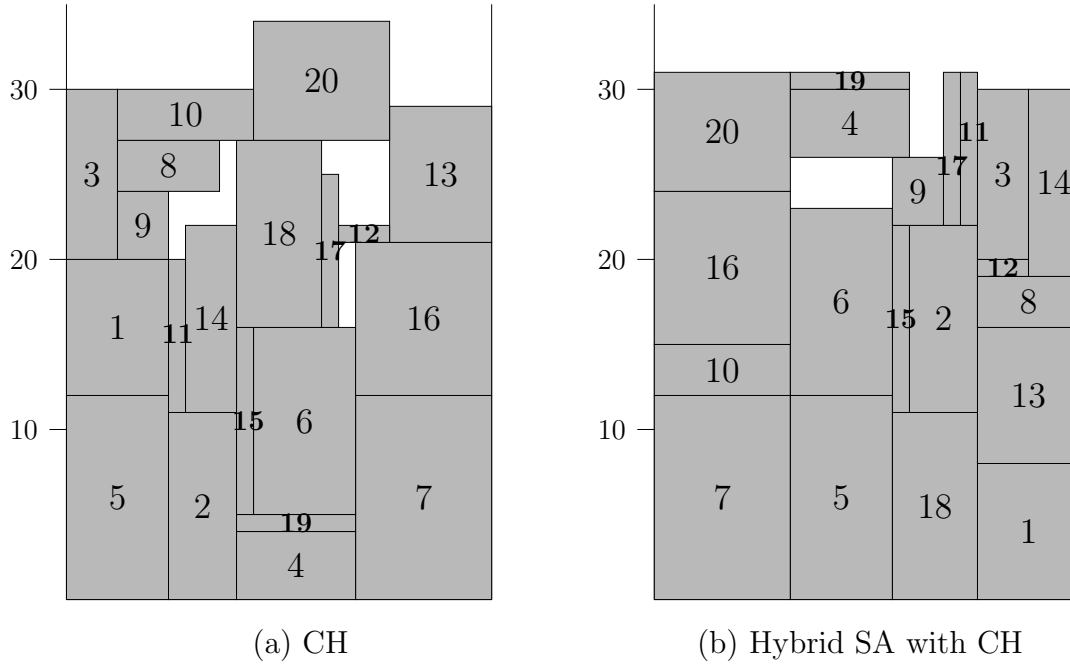


Figure 3.8: An example of a packing solution obtained with applying the CH algorithm to the list of items  $\mathcal{I}$  sorted by decreasing height is given by (a), while an example of a packing solution obtained when applying the hybrid SA combined with the CH algorithm as decoding routine to the list  $\mathcal{I}$  is given in (b). The resulting packing heights are 34 and 31 respectively.

### 3.3 Chapter Summary

This chapter served the purpose of introducing metaheuristic search techniques used in conjunction with heuristic algorithms to improve the solution quality and to generate sufficiently near-optimal solutions to the 2D SPP. This chapter opened with a discussion motivating the concept of metaheuristic techniques as general-purpose algorithms for combinatorial optimisation. An overview was provided that classifies these algorithms and also described the fundamental mechanisms and principles by which they work towards converging upon high quality solutions. In [Section 3.1](#) and [Section 3.2](#), two of the more popular search strategies, genetic algorithms and simulated annealing algorithms, were presented generally and thereafter described specifically for tackling the 2D SPP as hybrid approaches. Guidelines were offered in terms of selecting appropriate parameters in the GA and how to carefully construct the cooling schedule in SA. The fact remains that optimisation of the parameters is an open and challenging problem in and of itself and thus it is highly recommended that through trial and error these parameters are selected. In the course of the discussion, the problem instance of Bengtsson [8] was introduced and used as a working example to demonstrate the packed solutions of the

---

reviewed hybrid techniques compared to the heuristic techniques.

# **Part II**

## **Decision Support System**

---

# CHAPTER 4

---

## System Development

Decision-making is not an easy process. Often times, real-world problems are fraught with numerous variables and even more considerations, where small changes in one part of a system may lead to substantial consequences in another. Decision-making arises in many forms, but in most scenarios it consists of deriving the best possible option from a feasible set. It is not a simple task to be able to compare the results of different actions with respect to their desirability in decision problems. In the context of the strip packing problem, where items of varying sizes need to be placed into a strip, there are a plethora of decisions to be made on how to utilise the space or material efficiently while also recognising challenges related to cost and time availability. These factors underpin some of the motivation for a framework where problem instances can be readily solved and where high quality solutions can be produced in a reasonable space of time. It is possible to interpret the decision-making process of solving a strip packing problem in the framework of a computerised and user-friendly decision support system (DSS).

The goal of the DSS for a 2D SPP is that of deciding how to place items so that the minimum packing height of the items in the strip is obtained. This is achieved by leveraging the work contained in the literature for the problem and implementing the routines in such a way that the results are accessible to users or industry practitioners. Such a system should not only provide good solutions, but should also facilitate an understanding of the solution quality by means of aspects and metrics such as a visual display of the packing layout, the solution execution time, and the quality of a solution from one method compared to another.

Computerised systems are noted for their ability to offer *speedy computation*, *technical support*, and *quality support* [3, 66]. The first principle describes computers being able to perform repeated, rapid, and complex calculations, while the second refers to the ability to store, process, and retrieve information readily. The last aspect addresses its ability to handle and make evaluations for different scenarios or conditions. These qualities served as principles to guide the design and development of the DSS of this project titled **packR**.

## 4.1 The 2D Strip Packing DSS

*RStudio* [80] is a free and open-source software for data analysis and statistical computing based on the **R** programming language by the R Core Team [73]. It is one of the more frequently used integrated development environment (IDE) owing in great measure to its friendly graphical user interface that allows ease of use to a wide variety of applications. **R** has seen growing popularity in recent years [97]. By virtue of the fact that **R** is freely available under an open-source license, this has spurred the growth of its usage in numerous real-world problems. Additionally, the growth is driven by an engaged community of users and the result of which has led to the creation of a plethora of packages and functions available for usage [25]. The Comprehensive R Archive Network (CRAN) is a repository that contains more than 10,000 available packages for different types of applications (CRAN [22]).

One of the most important packages for the development of applications in **R** is the **Shiny** package [18] which was developed and is supported by the RStudio project. The application development armamentarium of **Shiny** lends itself to creating elegant and powerful web frameworks using **R**. **Shiny** has a relatively low barrier of entry in terms of ease of use thanks in large part to the fact that experience and domain knowledge in the use of HTML, CSS, and JavaScript are not required.

The high level architecture of any **Shiny** application is composed of two key **R** files – a server and a user interface (UI) file. The UI file serves as an interpreter for HTML – it allows for the creation of buttons, text fields, images, and other common HTML widgets. The server file controls how the different components of the application interface interact with one another taking in inputs and returning outputs. The combination of both these elements allows for the creation of a dynamic user interface [76].

It is in light of the aforementioned points, along with the authors' experience and familiarity with the **R** programming language, that the DSS of this project was developed using the **R** language in RStudio making extensive use of the functionalities of **Shiny**. The DSS was developed without the intention of being web-based, but rather for use by a user on a personal computer and falling under the desktop model-driven DSS classification of DSSs of Power [72].

*DesktopDeployR*<sup>1</sup> is a framework created by W. Lee Pang for deploying **Shiny** applications as self-contained **R**-based desktop applications. The framework relies on a portable **R** executable environment and a private package library to run the application. The application is run using the user's default web browser. The application is not dependant on any external software other than a web browser for displaying the application. This means that the DSS can be run on a computer that does not have **R** installed. The

---

<sup>1</sup><https://github.com/wleepang/DesktopDeployR>



DSS, **packR**, is available for download via GitHub<sup>2</sup>. Currently, only compatibility with the Google Chrome browser has been tested. The application folder structure is given in Figure 4.1.

Within the **packR** main folder are a number of sub-folders. The ‘shiny’ subfolder within the ‘app’ folder contains the **R** code for running the **Shiny** application (‘ui.R’, ‘server.R’, and ‘global.R’) as well as the **R** scripts for the implementation of the heuristic algorithms (‘heuristic.R’) and the hybrid GA and hybrid SA algorithms (‘hybrid\_ga.R’ and ‘hybrid\_sa.R’ respectively). The ‘app.R’ file found in the ‘app’ folder is used as a help function to execute the **Shiny** application. The benchmark instances as well as the user’s personal files are stored in ‘app/shiny/benchmark\_instances’. Packages listed in the ‘app/packages.txt’ file as well as all the necessary dependencies are stored privately in ‘app/library’. The ‘dist’ folder contains the portable **R** distribution. The ‘RUN\_APP.bat’ file found in the **packR** main directory is used to run the application.

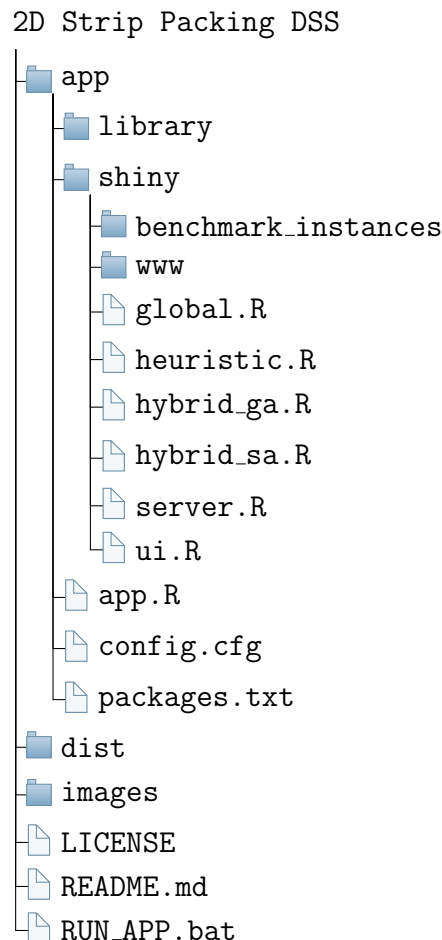


Figure 4.1: **packR** folder directory tree structure diagram.

<sup>2</sup><https://github.com/ahwallace/2d-strip-packing-dss>

## 4.2 Development and Implementation of the DSS

One of the important considerations important to the strip packing DSS is that of providing a user-friendly interface that allows the user or industry practitioners to solve SPP instances based on the various algorithmic implementations considered in this project, and the design of it was focused on it being accessible but also sufficiently interactive. There are three core components comprising the various elements and constituents of the DSS: (1) an *input component*, (2) a *processes component*, and (3) an *output component*. A model schematic of the components of the DSS and their interplay in solving SPP instances is presented in the form of a flow diagram in Figure 4.3 and the main steps are described and listed below.

- 1) The problem instance is loaded into the system. The DSS offers the functionality of the user selecting a pre-loaded benchmark instance or a personal instance which the user uploads into the system. Only *comma-separated values* (CSV) files are allowed to be uploaded to the DSS. For a problem instance of  $n$  items, the uploaded file needs to be structured, in a single column, as

$$\begin{aligned} &X,Y, \\ &i_1,h_1,w_1 \\ &i_2,h_2,w_2 \\ &\vdots \\ &i_n,h_n,w_n \end{aligned}$$

where the first line specifies the strip width ( $X$ ), the optimal packing height ( $Y$ ) which if unknown needs to be set to  $-999$ , and it terminates with a comma; the following  $n$  lines characterise the items of the instance, each item on its own line, where for the  $k^{th}$  item  $i_k$  is its reference ID,  $h_k$  its height, and  $w_k$  its width; all subsequent lines are left blank. All entries on each line are separated by commas. An example of an input instance file is provided in Figure 4.2. This images serves as a practical example to indicate the required format of the contents of this file and any deviations to this format renders the DSS unable to process the instance properly for packing. Furthermore, the values need to satisfy the criterion of being strictly positive with the exception of perhaps the known optimal height.

- 2) After the selection of the input data, the user decides on a heuristic placement policy. There are two possible heuristic routines and the user is required to select one to employ for the packing problem. These heuristic routines are the IBF algorithm of Wei *et al.* [93] and the CH algorithm of Leung *et al.* [53] detailed in this paper in Chapter 2 in Section 2.1.2 and Section 2.2 respectively, and whose implementation follows the methodology outlined by Algorithm 2 and Algorithm 3.
- 3) The user needs to decide on whether to incorporate a metaheuristic algorithm in generating a packed solution. If the decision is against this, i.e., only a heuristic

	A	B	C	D	E	F	G
1	250,-999,						
2	1,120,133						
3	2,135,186						
4	3,86,75						
5	4,103,73						
6	5,66,85						
7	6,135,97						
8	7,91,175						
9	8,131,176						
10	9,71,176						
11	10,153,72						
12	11,87,148						
13	12,168,107						
14	13,118,90						
15	14,140,109						
16	15,132,159						
17	16,152,93						
18	17,135,68						
19	18,121,158						
20	19,68,94						
21	20,155,76						
22							
23							

Figure 4.2: A screen shot of the contents of an Excel file specifying an example problem instance and the required format for the DSS. This instance comprises 20 items to pack in to a strip with width 250 and the optimal packing height is unknown.

algorithm is employed, then the user proceeds to specify an initial sorting order of the items in the instance before moving on to the commencing of a running of the algorithm in Step 5). The DSS offers support for sorting of the items in the orders of increasing or decreasing height, increasing or decreasing width, increasing or decreasing area, or otherwise simply defaulting to the initial order of the items in the instance as sorted by the item IDs.

If the user decides in favour of integrating a metaheuristic then the scheme of a hybrid metaheuristic approach is applied. This is the two-stage approach of a metaheuristic algorithm used to search for a good packing order and used in conjunction with a heuristic placement policy to produce a packing layout. The options available are that of the GA approach or the SA approach. The heuristic selected in the step previous is utilised.

- 4) This step arises if the user has selected to use a hybrid metaheuristic strategy. This step involves the user specifying the parameters with respect to the chosen metaheuristic.

In the case where GA is selected, the user is prompted to specify a population size, the type of selection for the reproduction phase, the crossover type and related probability, and finally the mutation probability. The DSS offers support for roulette wheel or tournament selection as the selection operator, whilst for the crossover operator there is OX or PMX functionality available. The algorithm was done with respect to the steps of [Algorithm 4](#). A point of interest here is that with regards to proportional selection, since the optimisation process involves the minimisation of packing heights, the fitness  $f_i$  of a chromosome in the population is defined as the inverse of the packing height,  $f(\pi) = 1/h(\pi)$ , where  $h(\pi)$  represents the packing height of the permutation  $\pi$  [\[56\]](#). Additionally, for selection for replacement, the strategy of Jakobs [\[46\]](#) in which the worst individuals in the current population, determined by their fitness, are replaced with the offspring generated by applying crossover and mutation operators if the fitness of the offspring is greater than the worst individual in the population was followed. The initial population is seeded with a number of good quality solutions. Namely, this includes the solution encodings of ordering by increasing and decreasing height, width, area, and perimeter. Therefore, this constituted 8 good quality solutions to the initial population being introduced. The initial population size cannot be set smaller than 10 in the DSS

If SA is selected, the set of parameters involving the initial temperature, the cooling schedule in which the cooling rate parameter is embedded, and lastly the epoch management policy need to be set. The cooling schedules available are the linear, geometric, logarithmic, and Lundy and Mees schedules of [Chapter 3 Section 3.2](#) specified by Equations [3.3 - 3.6](#). The implementation of this algorithm was done in a manner so as to be in line with the [Algorithm 5](#).

At the risk of the algorithms taking an unreasonable time to run, stopping criteria need also to be set for each metaheuristic. The DSS implements a maximum number of generations for the GA, and a maximum number of iterations and temperature lower bound for the SA.

- 5) At this stage the algorithm is tuned and ready to be run. One final consideration to make is to specify a maximum run time for the algorithm. This applies only to the metaheuristic approaches in conjunction with a heuristic placement routine. This is an additional stopping criterion inclusive of those criteria specific to each metaheuristic algorithm. The algorithm can begin packing!
- 6) The user receives the results of the algorithm execution. Available to the user in the output section include the total packing height; the packing layout in terms of

the bottom-left and top-right coordinates,  $(x, y)$ , for each of the rectangles with respect to the strip with the bottom-left corner of the strip initialised to  $(0, 0)$ ; a graphical display of the packed solution; the run time of the algorithm; and in the case of a metaheuristic algorithm a plot of the optimisation path of the algorithm.

A review of the literature and the implementation of packing algorithms cannot exist independent of the testing on known sample problems. In the strip packing literature, there exists a well-established set of benchmark instances, many of which with accompanying known optimal solutions. The instances of Burke *et al.* [15] and Bengtsson [8] were introduced in Chapter 2 and Chapter 3 respectively. These were pinnacle and enormously helpful in validating the implementation of the algorithms in code via a programming language. Appendix A contains a listing and an explanation for each of the benchmark packing instances employed in this project.

### 4.3 Chapter Summary

This chapter had the intention of introducing a DSS for the 2D SPP, and to discuss its implementation. The DSS incorporates the algorithms reviewed in Chapter 2 and Chapter 3. This chapter opened by describing the notions related to DSSs and also served to provide a brief motivation highlighting their importance as decision-making tools. Section 4.1 was dedicated to describing the practicalities of a 2D strip packing DSS based on the work of the earlier from the framework of its programming to its deployment, while Section 4.2 provided a high-level outline of its implementation. A flow diagram was presented in the discussion given by Figure 4.3 as a graphical representation of a skeleton structure for the workings of the various components of the DSS. The DSS, titled **packR**, is hosted on GitHub and is available for access and downloading from [90].

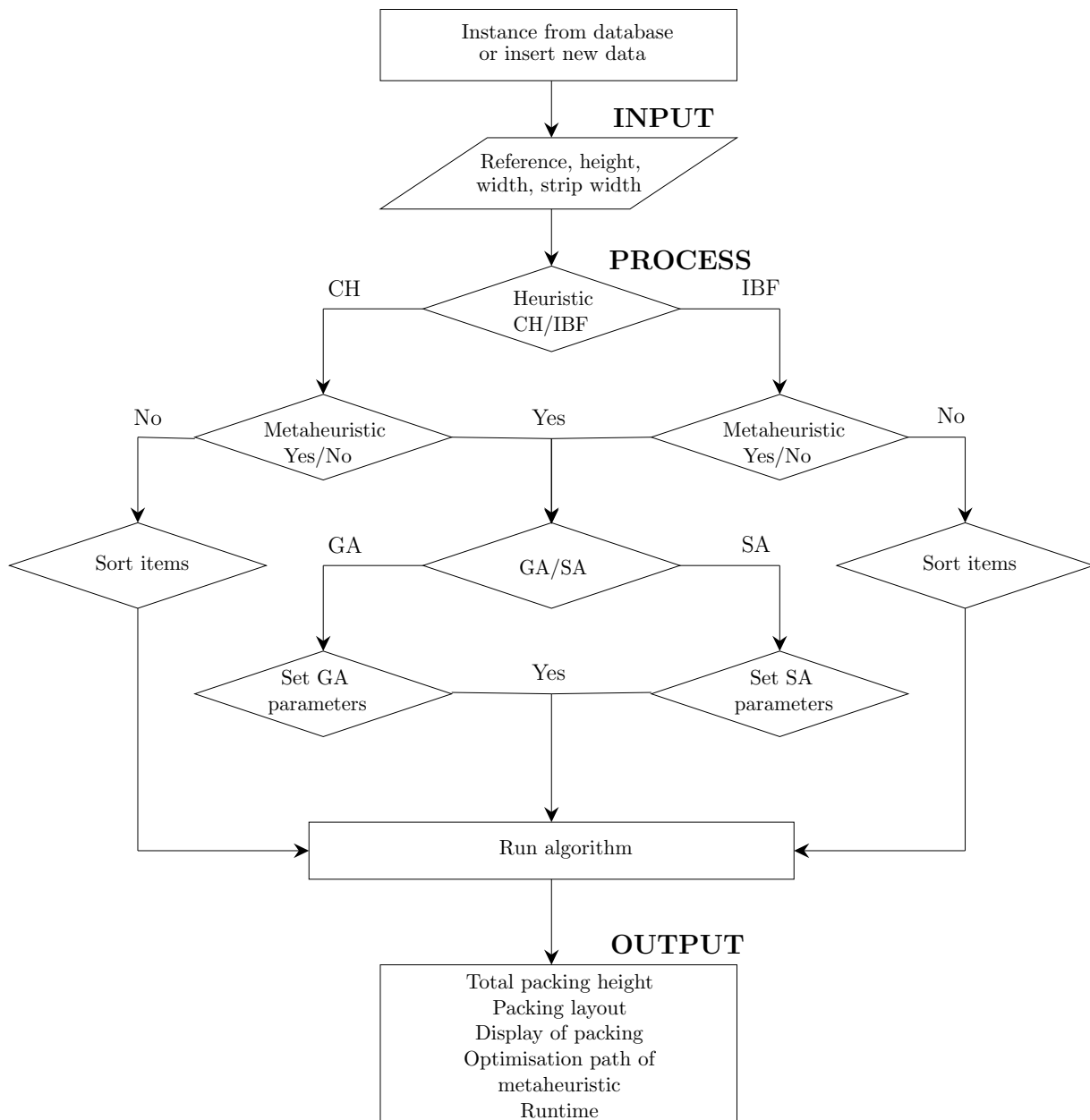


Figure 4.3: Flow diagram of the DSS input, process, and output components and their relations.

---

# CHAPTER 5

---

## System Interface

This section details the use of the **packR** DSS for running and comparing strip packing algorithms on user or benchmark instances. The sectional layout of the DSS is designed to be intuitive and allows the user to step through the application linearly. The DSS consists of four main tabs for navigation, namely, the *Instances*, *Algorithms*, *Compare*, and *About* tabs.

### 5.1 Tutorial

On initialisation of the DSS, the user is presented with a popup dialogue shown in Figure 5.1(a) introducing them to the application. The introduction dialogue invites the user to follow a short introductory tutorial to familiarise themselves with the various aspects of the DSS. The user is also given the option to skip the tutorial if they wish.

The tutorial is designed to give the user a guide to using the Instances section. It then instructs the user to move to the Algorithms section once a problem instance has been selected. The tutorial gives the user information regarding the algorithms selection procedure and outlines the necessary steps required before running the algorithm. A brief description of the output and where to find it is given. The tutorial does not give the user any information regarding the algorithms themselves, however additional information detailing the strip packing algorithms used is given in the About section. The user is then instructed on how the Compare section works. It specifies that the user is able to compare algorithms on the same problem instance and instructs the user to set the metaheuristic parameters in the previous Algorithms section.

Any additional information the user may require is placed in the About section. This includes a more detailed description of the algorithms and how they operate, a description of the data and an accompanying image of the data format and links to specific strip packing literature used in the DSS.

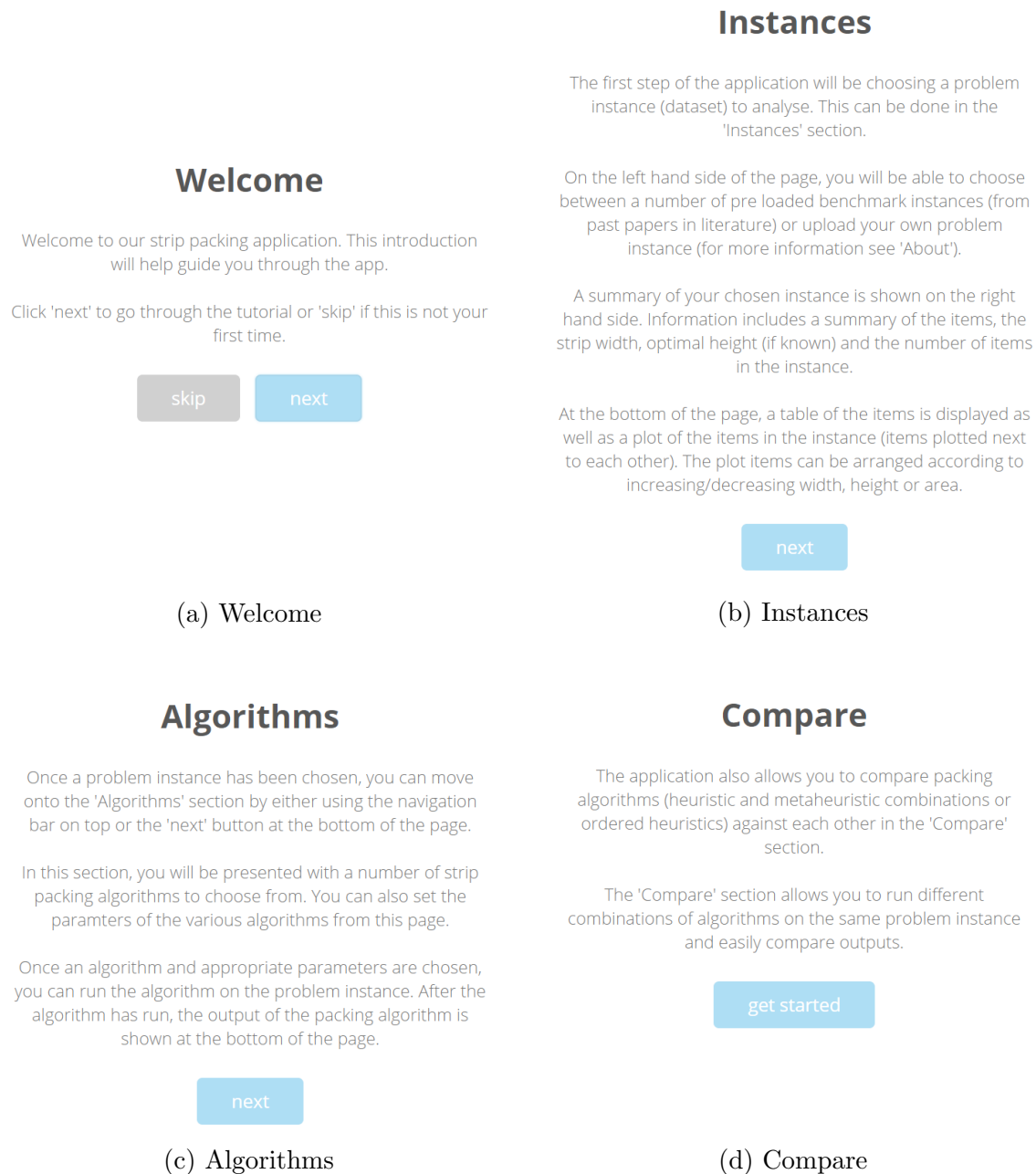


Figure 5.1: Introductory tutorial presented to the user upon initialisation of the DSS.



## 5.2 Instances

As outlined in [Chapter 4](#), the first decision a user is presented with is the problem instance to use. The user has one of two choices to make: upload their own problem instance or use one of the many pre-loaded benchmark instances. The Instances section interface is shown in [Figure 5.2](#). The layout of the Instances section allows the user to interact with the problem instances on the left-hand side of the screen, while information regarding the chosen problem instance is displayed on the right-hand side.

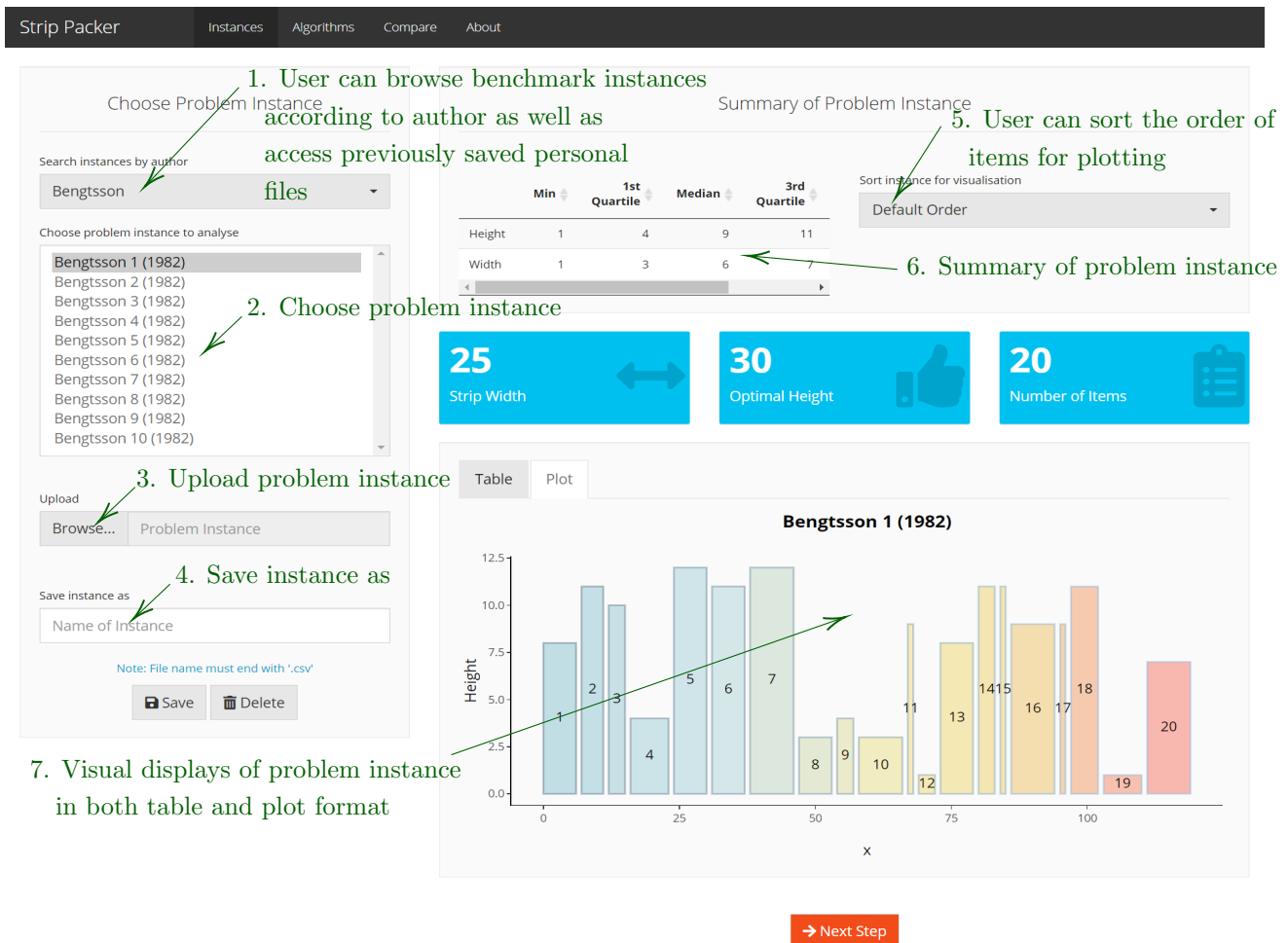


Figure 5.2: Screen shot of the Instances section of the DSS. Annotations are given in green.

### 5.2.1 Benchmark Instances

Users can scroll through a list of benchmark instances from various authors using the 'search instances by author' drop-down list (label 1 in [Figure 5.2](#)). The instances of the

selected author are displayed below for the user to choose. The benchmark instances and their respective authors are given in [Appendix A](#).

### 5.2.2 User Instances

If a user wishes to upload their own problem instance, they may do so by using the file upload button (label 3 in [Figure 5.2](#)). The user can name their file using the text input (label 4). It is important to note that the filename must end with the file extension, ‘.csv’ for the program to successfully load the new file. A number of checks are performed on user uploaded files to ensure that they are formatted correctly. Firstly, an error message is displayed if the file failed to load. If the file loaded correctly, the instance is checked to see if it contains a valid strip width and optimal height. The instance is also checked to see if the ‘Id’ column is correctly specified and the widths of the items are checked to make sure that they do not exceed the width of the strip. An error message is displayed conveying the relevant information to the user should any of the above errors occur.

User uploaded instances are saved under ‘Personal’ in the ‘search instances by author’ drop-down list.

### 5.2.3 Instances Summary

Information for a chosen problem instance is displayed on the right-hand side of the Instances section. Under the ‘Summary of Problem Instances’ header, the user is given a summary of the heights and widths of the items in the problem instance (label 6 in [Figure 5.2](#)). From this, the user can get an idea of the distribution of item heights and widths in the problem instance. Below the summary table, the strip width, the optimal height (if known, otherwise ‘Unknown’ is displayed), and the number of items in the problem instance are displayed in three information boxes. The user is shown a visual display of the problem instance in either a table format which displays the ‘Id’, ‘height’ and ‘width’ of the items or they can view a plot of the problem instance with the items plotted side by side (label 7 in [Figure 5.2](#)). Only instances with less than 50 items will be plotted, while only instances with less than 500 items will be displayed in a table. The plotted items can be arranged according to various criteria given in the drop-down list next to the summary table. (label 5 in [Figure 5.2](#))

## 5.3 Algorithms

Once a user has selected a problem instance to analyse, they will move to the Algorithms section (shown in [Figure 5.3](#)). The panel on the left of the screen is where the user is presented with a choice of heuristic and metaheuristic algorithms to choose from (label 1 in [Figure 5.3](#)). On the right, the user is shown an overview of the chosen heuristic and metaheuristic combination, and their chosen problem instance. The tabs at the top

(label 3) allow the user to input the required parameters for use with the genetic algorithm or simulated annealing algorithm. The output of the algorithms is also presented on the right-hand side of the screen, below the overview and parameter selection interface.

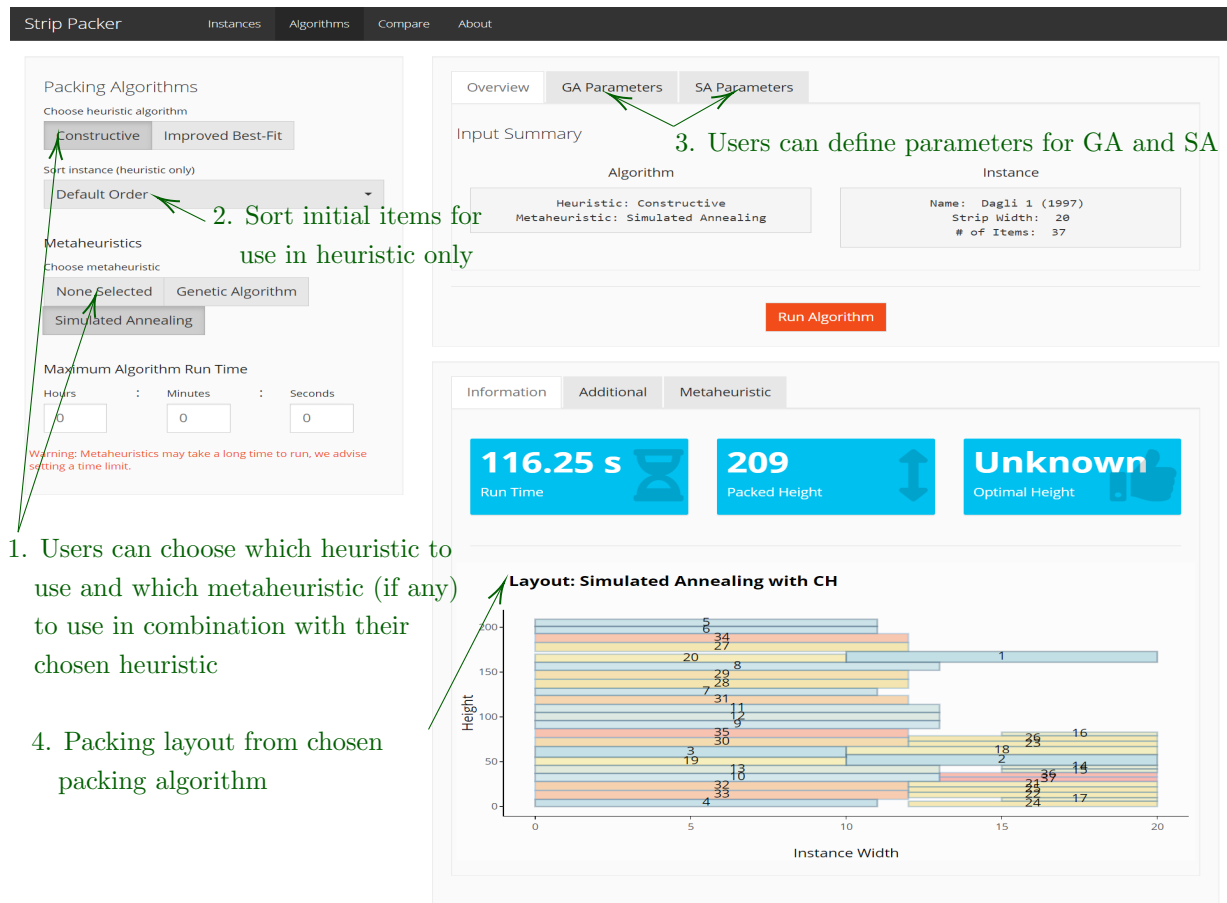


Figure 5.3: Screen shot of the Algorithms section of the DSS showing the packing layout of a problem instance by Dagli & Poshyanonda [23] found using the SA algorithm with the CH. Annotations are given in green.

### 5.3.1 Heuristics

The user has the choice between using the constructive heuristic or improved best-fit heuristic and can combine either one of these heuristics with a genetic algorithm or a simulated annealing algorithm. This gives the user 6 unique combinations of algorithms to apply to their problem instance. If a user only chooses to run a heuristic packing algorithm, then they can sort the initial order of the items using the drop-down list (label 2).

### 5.3.2 Metaheuristics

A user can select not to use a metaheuristic ('None Selected'), or they can choose to use 'Genetic Algorithm' or 'Simulated Annealing'.

If a user chooses to run the genetic algorithm, they are required to set the necessary parameters shown in Figure 5.4. All parameters have a default setting. These are determined based on general guidelines given in literature [99]. The default parameters allow the user to run the algorithm without having to change any of the parameter values. Although the use of default parameter values are in line with general recommendations from literature and will provide an adequate result most of the time, it is advised that the user fine-tune the default parameters to their specific needs.

Overview GA Parameters SA Parameters

Hybrid Genetic Algorithm Parameters

General Selection Crossover Mutation

Population size: 30

Number of generations: 50

Type: ☒ Roulette Wheel ☐ Tournament

Type: ☒ Partially Matched ☐ Order

Crossover probability: 0.7

Mutation probability: 0.01

Run Algorithm

Figure 5.4: GA parameters in the DSS.

Population size and number of generations are set using a numeric input field with only positive values being allowed. The user is given the choice of two different selection types, namely, proportional selection, implemented using the roulette wheel method and tournament selection. If a user chooses tournament selection, an additional parameter for tournament size must be set. This is displayed underneath 'Type' in the 'Selection' column only if tournament selection is chosen. The user is also given the choice between two crossover types, order crossover and partially matched crossover. Crossover probability is set using a slider with a range  $[0, 1]$ . Mutation probability is set using a slider with range  $[0, 0.1]$ . Although mutation probability can theoretically take on any value in the  $[0, 1]$  range, large values for mutation probability are not advised. Therefore, the range specified is adequate for allowing a diverse range of mutation probabilities.

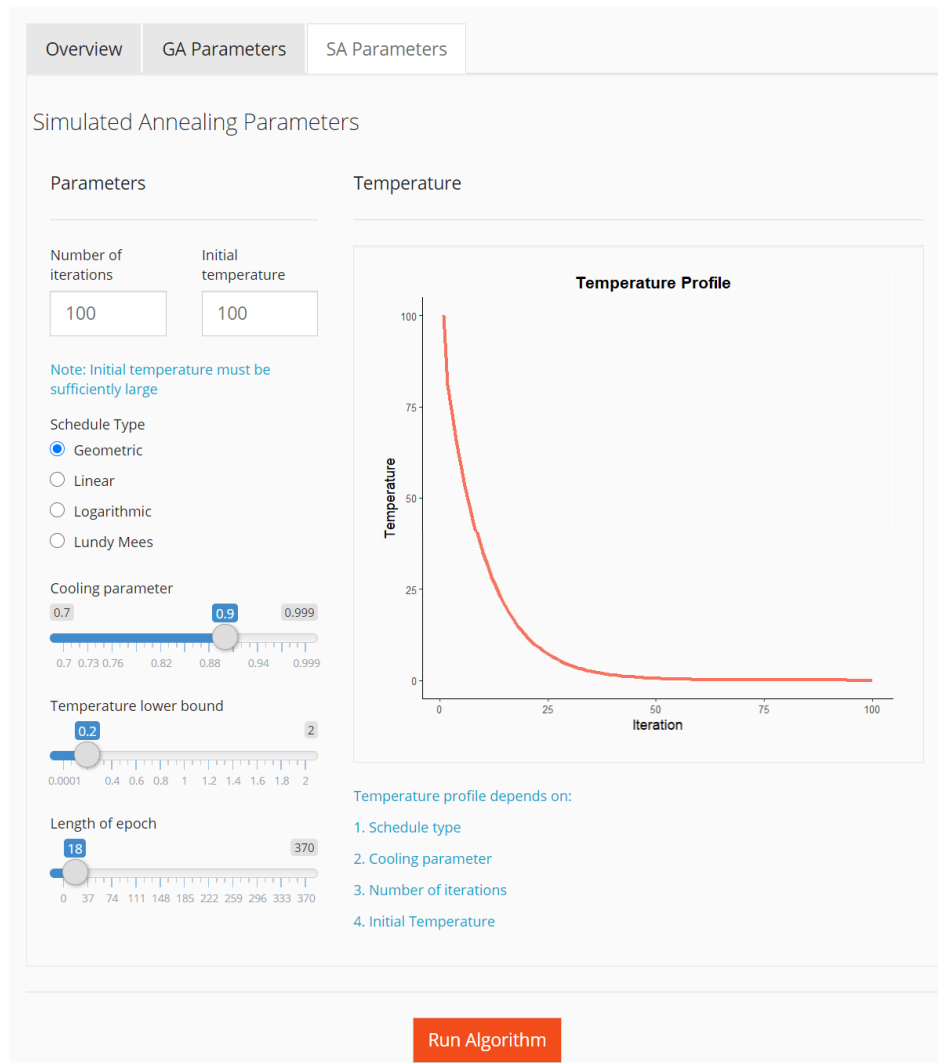


Figure 5.5: SA parameters in the DSS.

Simulated annealing parameters are shown in Figure 5.5. As well as being able to set various parameter values on the left, a dynamic plot of the temperature profile based on the values set for the cooling parameter, number of iterations, initial temperature, and the selected schedule type is shown to the right. The number of iterations and initial temperature are set using numeric inputs with default values of 100 for both. There are four schedule type options to choose from with the default being geometric as this is commonly used in strip packing problems [16, 43, 53]. The cooling parameter is chosen via a slider and the respective ranges follow the suggestions found in literature and as presented by the Equations 3.3 - 3.6 found in Section 3.2. The range of the cooling parameter as well as the default value depends on the chosen schedule type. The temperature lower bound varies and the default value depends on the initial temperature, similarly the length of epoch varies and the default value is dependent on the number of items in the chosen problem instance. There are no known methods to calculate the

cooling schedule or the recommended optimal parameter values for a variety of problems [14]; the values are often set through empirical experimentation and this principle served as a guideline for the setting of the ranges of the parameter values accepted. The DSS allows for sufficient experimentation of the parameters, but not beyond the boundary of being reasonable. For instance, a geometric cooling schedule with an embedded cooling rate parameter set to 1.1 is not allowed in the DSS considering this behaviour results in an increasing temperature profile nor would a negative initial temperature be permitted.

Due to the possibility of long run times for metaheuristics, a maximum run time can be set by providing the given hours, minutes, and seconds allowed for the algorithm to run.

### 5.3.3 Output

The algorithm output, found below the ‘Run Algorithm’ button consists of three tabs containing general information relating to the algorithm output, additional information regarding the algorithm layout, and results from the optimisation of the GA and SA metaheuristics.

The ‘Information’ tab contains information boxes displaying the run time of the algorithm and the height of the packed items. The right-hand side information box gives the optimal height of the instance for easy comparison with the packing solution height. Below the information boxes is a plot of the packing layout. The plot is built using the **R** package `plotly` [83] which allows the user to interact with the display. With `plotly`, the user can hover over an item in the packing layout to display the item’s ID, height, and width. A useful function provided by `plotly` is the ability to zoom in on the packing layout. This is a useful feature for layouts with a large number of items. Using `plotly` also allows the user to download an image of the plotting layout to their device.

The ‘Additional’ tab displays the packing layout in table format for the user to inspect, should they wish. Output shown in ‘Information’ and ‘Additional’ is available when using heuristics only and when using heuristics in combination with a metaheuristic algorithm. The ‘Metaheuristic’ tab contains information regarding the running of the metaheuristics. The stopping criterion of the algorithm that has been met is displayed above plots of the optimisation paths of the respective metaheuristic. For the genetic algorithm, two optimisation paths are displayed, the best packing height found in each generation and the average packing height of the population for each generation. For simulated annealing, the fitness progress (measured as the average packing height of solutions at each temperature point) is displayed. An example of the ‘Metaheuristic’ tab output for simulated annealing is shown in Figure 5.6.

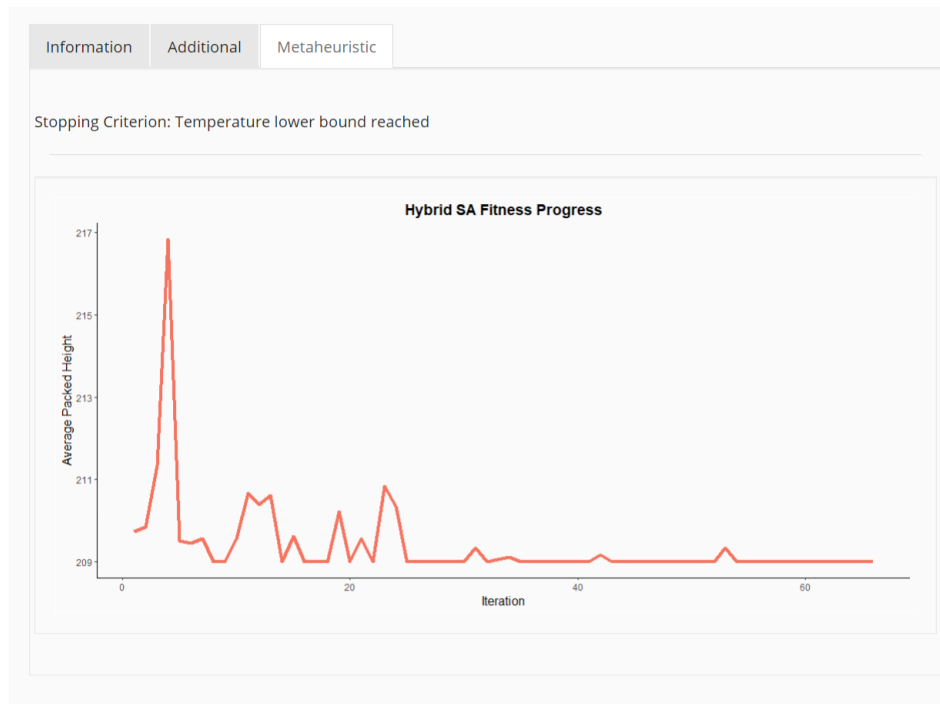


Figure 5.6: Example ‘Metaheuristic’ output for simulated annealing showing the stopping criterion and fitness progress.

## 5.4 Comparison

The comparison section allows the user to easily compare two or three algorithms against each other.

The Compare section shown in Figure 5.7 is divided into columns (‘Slots’) based on the number of algorithms being compared. For each Slot, the user can select a heuristic and the initial order or they can instead select a combination of a heuristic and metaheuristic algorithm. The 2 heuristics and 2 metaheuristics in combination give 4 unique combinations while the 2 heuristics combined with the 6 available sorting orders give 12 unique combinations. Adding the 2 heuristics sorted in their default order to the previous two sets combine to give 18 unique combinations to choose from in the Compare section.

The algorithms are all run on the same problem instance, making the various algorithm outputs directly comparable. If a user chooses to compare a metaheuristic, the parameters for their chosen metaheuristic are inherited from the previous Algorithms section. This is restrictive as it does not allow the user to compare the same metaheuristic with different parameter values. A maximum run time can be set that applies to all metaheuristic algorithms being compared.

The screenshot shows the 'Compare' section of the DSS interface. At the top, there are tabs for 'Strip Packer', 'Instances', 'Algorithms', 'Compare', and 'About'. The 'Compare' tab is active. Below the tabs, there are three columns for selecting algorithms. The first column has 'Heuristic: Constructive', 'Sort instance (heuristic only): Default Order', and 'Metaheuristic: None Selected'. The second column has 'Heuristic: Improved Best-Fit', 'Sort instance (heuristic only): Default Order', and 'Metaheuristic: Simulated Annealing'. The third column has 'Heuristic: Constructive', 'Sort instance (heuristic only): Default Order', and 'Metaheuristic: Genetic Algorithm'. Above these columns, there is a 'Number of Algorithms to Compare' section with 'Two' and 'Three' buttons. A red 'Compare' button is located below the columns. Below the 'Compare' button, there is a 'Maximum Run Time Per Algorithm' section with input fields for 'Hours' (0), 'Minutes' (1), and 'Seconds' (0). Below this, there is a 'Note: Metaheuristic parameters inherited from 'Algorithms' section'. Below the note, there are three boxes showing the output for each algorithm. The first box shows 'Heuristic: Constructive', 'Initial Order: Default Order', 'Metaheuristic: None Selected', 'Run Time: 0.05 s', 'Packed Height: 36', and 'Stopping Criteria: NULL'. The second box shows 'Heuristic: Improved Best-Fit', 'Initial Order: Default Order', 'Metaheuristic: Simulated Annealing', 'Run Time: 60.78 s', 'Packed Height: 31', and 'Stopping Criteria: Time limit reached'. The third box shows 'Heuristic: Constructive', 'Initial Order: Default Order', 'Metaheuristic: Genetic Algorithm', 'Run Time: 61.08 s', 'Packed Height: 31', and 'Stopping Criteria: Time limit reached'. Annotations in green text and arrows point to various parts of the interface: '1. User can choose how many algorithms to compare' points to the 'Number of Algorithms to Compare' section; '2. Parameters for GA and SA must be set in the Algorithms section' points to the 'Metaheuristic' dropdowns; '3. Maximum run time for each algorithm' points to the 'Maximum Run Time Per Algorithm' section; and '4. Output for each algorithm given side by side for easier comparison' points to the three output boxes.

Figure 5.7: Screen shot of the Compare section of the DSS showing the comparison of three algorithms. Annotations are given in green.

The output of the comparison section (label 4 in Figure 5.7) includes the algorithms and ordering used, the run time and packing height of the algorithms and the stopping criterion if a metaheuristic was used. Plots of the algorithms' packing layouts are also given as shown in Figure 5.8. Each algorithm packing layout plot is given its own 'Slot' corresponding to the order of the algorithms chosen for comparison.

## 5.5 About

The About section as shown in Figure 5.9 gives the user some additional background information regarding strip packing along with the heuristic and metaheuristic algorithms used in the DSS. Should the user encounter a problem while using the DSS, the About section can be consulted where brief help descriptions are provided. The data format as well as an image of the raw instance in '.csv' format is given for the users that upload their own problem instances. References to the papers of Wei *et al.* [94] and Leung *et al.* [53] for both of the heuristic algorithms as well as several other strip packing literature integral in the development of **packR** are also provided [15, 44, 74].



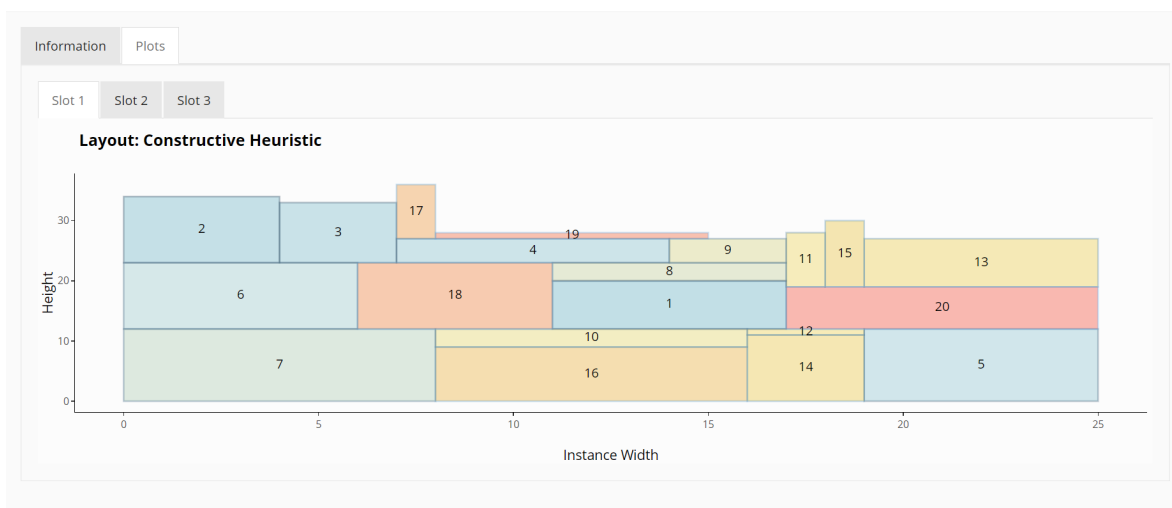


Figure 5.8: Packing layout plot for the first algorithm being compared.

## 5.6 Chapter Summary

The graphical user interface of **packR** was presented in this chapter by means of various screen shots. This chapter mimicked the path the user would navigate through while using the software. Starting with a tutorial, [Section 5.1](#) presented the display of the DSS upon initialisation of the application. The DSS comprises four main tabs (*Instances*, *Algorithms*, *Compare*, and *About*) for navigating through the application and the sections of [Section 5.2](#) through to [Section 5.5](#) were devoted to describe their respective interfaces in detail and to highlight the various functionalities available to the user in each tab. Examples of the *input*, *processes* and *output* components of [Chapter 4](#) were shown in this chapter in the DSS software.

2D Strip Packing DSS

Instances Algorithms Compare About

Help

The application is structured so that a user moves through the application sequentially.

The user starts by either selecting or uploading data in the 'Instances' section. Once an instance has been chosen, they can move onto the 'Algorithms' section where they are given the choice either a heuristic algorithm or a combination of heuristic and metaheuristic to use on their chosen problem instance. If a user selects a metaheuristic, they will need to input the required parameters.

Once a user has chosen the algorithms and parameters, they can run the algorithm on the selected instance. A plot of the packing layout as well as other information such as the height of the packing solution is shown to the user

A user may also wish to compare different algorithms against each other on their chosen instance in the 'Compare' section.

Data

Data uploaded by users must have the following structure:

1. First line: width of the strip, optimal height (if optimal height is unknown - optimal height value must equal -999)
2. Other lines: reference, height, width
3. Data must be in .csv format

	A	B	C
1	40,-999,		
2	1,8,6		
3	2,11,4		
4	3,10,3		
5	4,4,7		
6	5,12,6		
7	6,11,6		
8	7,12,8		
9	8,3,6		
10	9,4,3		
11	10,3,8		
12	11,9,1		
13	12,1,3		
14	13,8,6		
15	14,11,3		
16	15,11,1		
17	16,9,8		
18	17,9,1		
19	18,11,5		
20	19,1,7		
21	20,7,8		
4	1982Bengtsson6		

Overview

Strip packing is a complex combinatorial problem with a lot of real world applications. This software focuses on orthogonal, fixed orientation, two-dimensional strip packing.

The aim of this application is to assist researchers in studying various heuristic and metaheuristic implementations.

Heuristic Algorithms

Heuristic algorithms are simple rules based packing algorithms that allow users to quickly obtain good quality (although not necessarily optimal) solutions to packing problems.

Constructive heuristic scoring rule for the case when  $h_1 \geq h_2$ :

Improved Best-Fit heuristic scoring rule:

Both the Constructive and Improved Best-Fit heuristics rely on a two-stage packing procedure whereby a suitable space is found and thereafter an item to fit into the space is chosen based on the above mentioned scoring rules. The item with the highest score for the chosen space will be placed.

Both methods rely on the bottom-left criteria where an item will be placed in the lowest, left most available space. If no item can fit into the bottom-left space, the next lowest, left-most space is evauted until all items are packed.

Items in the the Constructive and Improved Best-Fit heuristics are placed next to the tallest neighbour. That is, the item is placed on the side of the space with the largest height.

Metaheuristic Algorithms

Metaheuristic algorithms combine a heuristics algorithm with a search for the best initial order of items. The metaheuristic searches a set of item permutations while the heuristic algorithm is used to decode the item permutations into packing solutions. The best packing solution of the heuristic algorithm given the initial ordering of items found by the metaheuristic is returned.

Hybrid Genetic Algorithm

Is based on the process of natural selection where an initial population of solutions taking the form of packing permutations are evolved over time using the genetics-inspired operators of crossover and mutation.

Hybrid Simulated Annealing

Is based on the physical process of annealing. In broad terms, the simulated annealing (SA) algorithm typically commences with an initialisation of the starting parameters - a random initial solution and a relatively largestarting temperature. At each iteration, a neighbour of the current solution is randomly generated by some suitably defined mechanism. The objective function is evaluated atthis new point and thereafter the decision to accept or reject this as the new solution is performed probabilistically. Whilst running through the SA algorithm iteratively the temperature is cooled.

Acknowledgments

This software was developed by Aidan Wallace and Leslie Wu, under the supervision of Dr. R. Georgina Rakotonirainy, in partial fulfilment of an Honours in Statistical Sciences from the University of Cape Town.

References

1. E. K. Burke, G. Kendall, and G. Whitwell, A new placement heuristic for the orthogonal stock-cutting problem, *Operations Research*, 52 (2004), pp. 655-671
4. E. Hopper and B. C. Turton, A review of the application of meta-heuristic algorithms to 2d strip packing problems, *Artificial Intelligence Review*, 16 (2001), pp. 257-300.
3. S. C. Leung, D. Zhang, and K. M. Sim, A two-stage intelligent search algorithm for the two-dimensional strip packing problem, *European Journal of Operational Research*, 215 (2011), pp. 57-69.
5. R. G. Rakotonirainy, Metaheuristic solution of the two-dimensional strip packing problem, PhD thesis, Stellenbosch University, Stellenbosch, 2018.
2. L. Wei, Q. Hu, S. C. Leung, and N. Zhang, An improved skyline based heuristic for the 2d strip packing problem and its efficient implementation, *Computers & Operations Research*, 80 (2017), pp. 113-127.

Figure 5.9: Screen shot of the About section of the DSS.

# **Part III**

## **Conclusion**

---

---

# CHAPTER 6

---

## Dissertation Summary

### 6.1 Summary of Dissertation Contents

Packing problems have widespread application in business and industry, so much so that there has been an enormous push to further the knowledge on understanding optimal packing layouts. Júnior [47] claims that “the reduction of raw material waste during cutting or packing operations represent a cost reduction in different industrial sectors.” This project set out to survey some of the influential and important groundwork in the literature of two-dimensional strip packing algorithms. Thereafter, an exploration of the most recent solution approaches were detailed. Based on this work, a computerised decision support system was developed with the primary objective of providing users and industry practitioners alike a user-friendly platform for solving their problems.

In [Chapter 1](#), the basic notions and related terminology in the context of the strip packing problem were covered. The methods employed for these packing problems belonged to the classes of approximate techniques: heuristic algorithms and hybrid metaheuristic algorithms, and a general overview describing these algorithms as well as their application in the context of the strip packing problem was provided in [Chapter 2](#) and [Chapter 3](#) respectively. In the first of these two chapters, a survey and review of two selected heuristic methods were performed, namely, the *best-fit algorithm* of Burke *et al.* [15], along with the more recent revision of the *improved best-fit algorithm* of Wei *et al.*, and the *constructive heuristic* of Leung *et al.* [53]. These methods define a set of rules for packing items into the strip. [Chapter 3](#) introduced metaheuristic algorithms as search methods to determine an optimal ordering of the items to be decoded by a heuristic algorithm, the latter used as a placement routine. The popular methods of a hybrid *simulated annealing* and a hybrid *genetic algorithms* approach to the strip packing in the literature was detailed.

All of these above algorithms were implemented in the **R** programming language and a user interface that allowed for accessible use of these algorithms to solve benchmark instances and personal instances as well as facilitating algorithm comparisons was devel-

oped in Shiny. The *DesktopDeployR* framework of W. Lee Pang [70] allowed for the application to be distributed as a standalone desktop application. The decision support system, titled **packR**, along with all the relevant code is available to be downloaded from the online GitHub repository at [90]. Motivation for decision support systems as effective decision-making tools was described, and a detailing of the development process of this project was given in Chapter 4, while in Chapter 5 a demonstration of the user interface components and design was provided.

It is the authors' opinion that the initial aims and objectives identified in Chapter 1 have been satisfactorily realised. Moreover, it is hoped that the work of this paper and the accompanying decision support system is a valuable addition to the strip packing literature, and the more general cutting and packing problem.

---

---

# CHAPTER 7

---

## Future Work

Packing problems have been the subject of extensive research and the literature is expansive; however, there remains still much more work to be done with regards to finding efficient algorithms and techniques, capable of handling a variety of SPP types than those covered in this project, that are not computationally expensive. While on the whole this project was able to meet its objectives satisfactorily, it was during the course of performing the necessary research and implementing the algorithms studied that the authors identified numerous aspects specific to the context and aims of this project that could be furthered investigated and improved upon. The authors recognise these aspects as being important, relevant and having the potential to add immense value and to complement the work carried out in this project. Some of these aspects are outlined in this chapter, and they serve as suggestions for possible avenues for any follow-up study done in the future.

### 7.1 Alternative SPP Solution Techniques

#### Heuristic algorithms

In this project, only plane algorithms in the SPP heuristics typology were reviewed and implemented. In [Chapter 2](#), algorithms of the class of level and pseudolevel types were alluded to. As mentioned, these are algorithms which are based on having rectangles be packed in horizontal levels drawn across the strip. The bottom of the strip constitutes the bottom level of the strip, and each subsequent level is given by the height of the tallest rectangle placed on the level previous. While level algorithms impose the constraint that the bottom edge of each item in a level be in contact with the level boundary, pseudolevel algorithms specify a relaxed version where the contact of the bottom edge of the level is not required.

Ortmann and Van Vuuren [\[69\]](#) performed a study where they compared the performance of strip packing algorithms belonging to the different classes when applied to 1170 benchmark instances from various authors. The result of their study was the conclusion

that some algorithms performed better than others in terms of solution quality, and that the execution time of some algorithms were noticeably worse than others. These both are aspects that can be furthered explored, and the framework of a DSS can be used to facilitate such a study.

## Metaheuristic algorithms

The metaheuristic methods studied and employed in this project are based on the more popular methods for solving the 2D SPP. However, numerous researchers have explored other alternative metaheuristic search techniques for the 2D SPP. In the class of trajectory-based methods, SA being an example of such a method, the tabu search method of Glover [35] has been applied to study the SPP. The work of Gómez-Villouta *et al.* [37] and Hamiez *et al.* have used this method and there is a lot promise and as such warrants further studying. In the paper of Gómez-Villouta *et al.*, they proposed the *Consistent Tabu Search* algorithm for a 2D SPP and its performance was assessed on 21 well-known hard instances. Computational experiments were able to reach the optimal for 9 of the problem instances.

In the class of population-based methods, the method of ant colony optimisation of Dorigo [27] has been a point of study for the 2D SPP in the literature. Briefly, the ant algorithms try replicate foraging behaviour of social ants. Using this metaphor, the pheromone of ants, chemicals used for communication, and their concentrations are compared to optimisation routes and paths where for higher pheromone concentrations the solution quality is better. Salto *et al.* [81, 82] have used this concept to propose a hybrid ant colony system to tackle the 2D SPP, which they call the *Ant Colony System*. When used in conjunction with a fine-tuned local search procedure, they generated results that could compete with genetic algorithm methods in terms of solution quality and which also exhibited low execution times.

Using only the example of tabu search and ant colony methods above, there is clearly much to be learnt and insight to be gained from adopting different perspectives in considering the 2D SPP.

## Devise novel and/or improve upon existing approaches

It is after analysing various heuristic packing layouts and noticing apparent shortcomings in specific heuristic scoring rules for given item sets that we attempt to contribute our own improved heuristic scoring rule.

## Improving the Constructive Heuristic

Using the constructive heuristic of Leung *et al.* [53] as our basis, we propose a set of possibly improved constructive heuristic scoring rules. The table of scoring rules for our

improved constructive heuristic (ICH) is given in Table 7.1.

Table 7.1: Scoring rules for evaluating items and calculating change in number of available spaces  $s$  for the proposed improved constructive heuristic

Case	Condition	$f$	$m$	Side
$w = width[i]$	$h_1 \ \& \ h_2 = length[i]$	5	-2	-
	$h_1 \ \text{or} \ h_2 = length[i]$	4	-1	-
	$h_1 \ \& \ h_2 < length[i]$	3	0	-
	$h_1 \ \text{or} \ h_2 > length[i]$	2	0	-
$w > width[i]$	$h_1 = length[i]$	1	0	Right
	$h_2 = length[i]$	1	0	Left
and $h_1 \geq h_2$	$h_1 \ \text{and} \ h_2 \neq length[i]$	0	+1	Left
and $h_1 < h_2$	$h_2 \ \text{and} \ h_1 \neq length[i]$	0	+1	Right

The algorithm uses the same skyline representation of the packing space and searches for the lowest, leftmost available space to place an item. Item scores  $f$  range from 0 to 5. The item with the highest score will be placed.  $m$  refers to the change in the number of available packing spaces due to the placement of an item and the side of the chosen space an item is placed is also determined. The side an item is placed if the items width ( $width[i]$ ) is equal to the width of the space ( $w$ ) is not important. If the items width is less than the space width, the side of the space an item is packed on becomes an important consideration. Should the height of the item ( $length[i]$ ) match either one of the heights of spaces sides ( $h_1 \ \& \ h_2$ ), then the item is placed on the side of the matching height. If the item's height does not match any of the sides' heights, then the item is placed on the left or right of the space using the tallest neighbour criterion.

Benchmark instances are useful towards evaluating the performance of algorithmic procedures, in fact this is the standard approach. Extensive work will be required towards the appraisal of this proposed ICH with respect to space and time efficiencies, as well as the solution qualities as compared to existing methods. However, a cursory evaluation of the performance of the ICH compared to the CH in terms of packing heights for various benchmark instances is provided in Table 7.2. Additionally, application of the ICH as applied to packing the item sets  $\mathcal{H}$  and  $\mathcal{I}$  introduced in Chapter 2 and Chapter 3 is presented in Figure 7.1.



Table 7.2: Packing heights of the ICH as compared to the CH for various benchmark instances. The columns labelled ‘Strictly Less’, ‘Equal’, and ‘Strictly Greater’ contain the number of the packing instances where the packing heights of the ICH performed better, equal to, and worse than the CH respectively when packing the items sorted initially in the default order of the instance.

Benchmark Author	Strictly Less	Equal	Strictly Greater	Total
Bengtsson	4	2	4	10
Jakobs	1	0	1	2
Dagli & Poshyanonda	0	1	0	1
Ratanapan & Dagli (1997)	0	1	2	3
Ratanapan & Dagli (1998)	0	0	1	1
Babu	1	0	0	1
Burke & Kendall	1	0	0	1
Hifi SCPL	3	6	0	9
Hopper (C)	10	4	7	21
Hopper (NT)	20	15	35	70
Wang & Valenzuela (Nice)	43	157	39	239
Burke, Kendall & Whitwell	5	1	6	12
Total	94	188	100	382

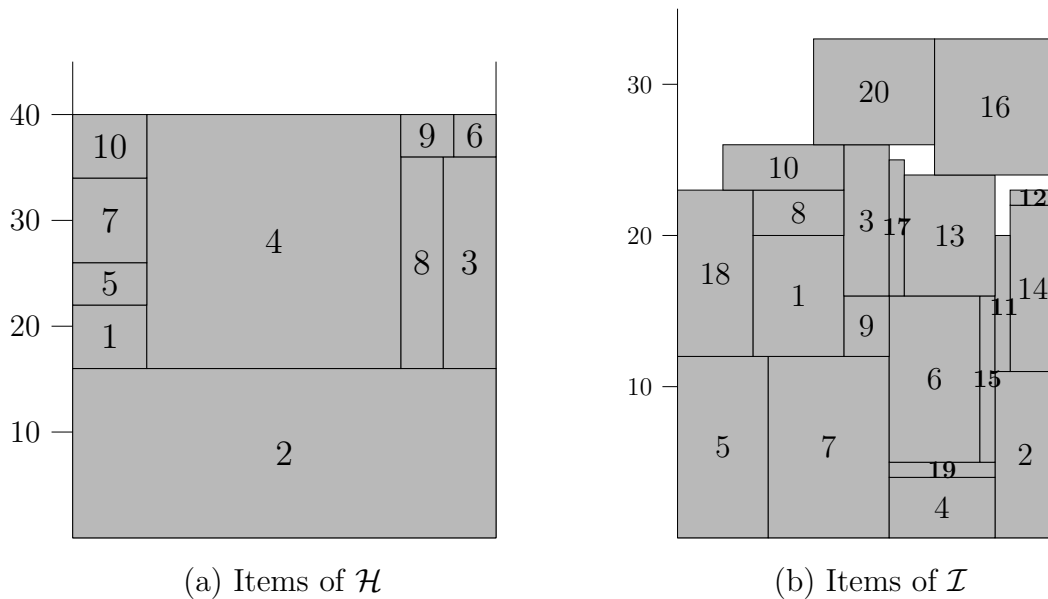


Figure 7.1: The packing solutions obtained with applying the ICH algorithm to the list of items of  $\mathcal{H}$  and  $\mathcal{I}$  both initially sorted by decreasing height. The resulting packing heights are 40 and 33 respectively.

## 7.2 Improvements to the DSS

### Convert algorithm implementation to a lower level language

The backbone of the DSS was coded in the **R** programming language. This was in large attributed to the authors' experience and comfort with the language. While **R** is efficient and popular for the purposes of data analysis and providing exemplary support for data wrangling and graphic and data visualisation, these aspects were not particularly relevant in the algorithmic implementations. As is, the execution times of the metaheuristic algorithms, for instance, leave plenty to be desired especially when scaling the problem instance size.

It is recommended that perhaps the code be converted to a lower level language. For example, using the programming language **C++** as a point of reference, **R** is largely interpreted while **C++** is compiled; **R** objects are stored in RAM and without the aid of large and powerful computing machines **R** is inefficient for handling large problem instances or even storing the information required while iterating through the hybrid metaheuristics in solving these instances, while **C++** has superior memory management.

Alternatively, parallelisation of code may be explored to improve algorithm run times.

### Additional features and functionality

In the designing of the DSS of this project, there were numerous instances of *what-if?*. Many of these scenarios were embedded in the accessibility of the application. Example scenarios included different manners in which personal problem instances could be introduced in to the software (via a drawing pad, for instance) or incorporating existing problem generators such as that of Silva *et al.* [85] titled *2DCPackGen*, allowing the user to set unique parameters for metaheuristics in the Compare section such that the user can compare the same algorithm but with different parameter values against each other, and dynamic and animated plots of the packing layout and fitness progression while the algorithms are being executed, to name a few. An incorporation of any of the above would create a richer and more enhanced user experience of the software.

### Further testing to identify and fix bugs

Time constraints meant that testing of the DSS was limited. A priority was placed on ensuring the back-end was functioning as required; that is to say functionality outweighed form and it was preferred that the DSS relate to its primary intended purpose. It is suggested that substantial testing of boundary cases as well as improvements on the front of exception handling be done.

---

# APPENDIX A

---

## Benchmark Instances

The implementation of the algorithms in this project required extensive testing and validation using numerous datasets and problem instances from various repositories in the literature. Information regarding the instances that were employed and used for the evaluations are presented below. Furthermore, these benchmark instances are loaded as example problem instances in the DSS available for the user to select, examine, and solve. A brief description of the characteristics of each set of instances is contained here and this material borrows largely from the work compiled by Rakotonirainy [74]. The instances are presented in chronological order. Unless specified otherwise, the data instances may be accessed and downloaded from the online repository published by Van Vuuren and Ortmann [89].

### The Instances of Bengtsson (beng)

Bengtsson's (1982) [8] set of instances (beng) consists of ten packing problems each with varying number of items. The first five concern a packing into a strip of width 25, while the remaining five are for a width of 40. The optimal packing heights are known for several of the instances, while for the remaining instances Martello *et al.* [61] have determined lower bounds. The rectangular items were generated by assigning to them the lengths  $12r + 1$  and widths  $8r + 1$  both rounded to the nearest integer value and where  $r$  is a random uniform number on the range  $(0, 1)$ .

The problem instance example used as an illustrative tool of the algorithms for Chapter 3, referred to as  $\mathcal{I}$ , is an instance of Bengtsson.

### The Instances of Jakobs (J)

Jakobs (1996) [46] generated two SPP benchmark instances (J). These were constructed by cutting a stock rectangle of height 15 and width 40 randomly into smaller pieces. The first instance consists of 25 rectangles while the second 50 rectangles. The optimal solutions for both are known and are not necessarily guillotineable.

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

Dagli and Poshyanonda (1997) [23] and Ratanapan and Dagli (1997-1998) [77] generated

four problem instances (DP), but provided no details as to their construction. The number of items to pack and the strips from one instance to another vary. Optimal solutions are not known for these benchmark instances. These data instances can be obtained through the EURO Special Interest Group on Cutting and Packing (ESICUP) repository available at [30].

### The Instances of Hifi (SCPL)

Hifi (1999) [40] proposed 9 problem instances (SCPL) that were used in his study to assess the relative performances of his packing algorithms. There are no details as to how they were constructed except that they were randomly generated. Optimal solutions are not known for any instance in the set. The benchmark instances are relatively small and each instance varies from one to the next with respect to strip width. They are ordered by increasing complexity. These data sets are accessible from [41]

### The Instances of Babu (babu)

Babu and Babu (1999) [4] produced a single SPP instance (babu) that was used for the evaluation of the performance of their packing approach. Other than the items being cut from an initial large sheet, there are few details regarding the manner in which this instance was generated. There are 50 items to pack into a strip with width 1000 and the optimal packing is known. This instance can be sourced from the online repository of Wei and Zhu at [96].

### The Instances of Burke, Kendall and Whitwell (BK, N)

Burke and Kendall's (1999) [14] proposed a single instance (BK) and is drawn from the paper of Christofides and Whitlock [19]. The instance arises from Figure 4 in the latter paper and by multiplying all item dimensions by 2. This instance consists of 13 items to pack in a strip with width 80. The optimal height is known.

Burke, Kendall and Whitwell (2004) [15] used a benchmark generating algorithm that used the methodology of making random horizontal or vertical cuts to randomly selected rectangles starting with a large rectangle (thus the optimal solution is known). The cuts are made in a manner such that pre-specified minimum dimension constraints are not violated. The process iterates until the number of required rectangles are produced. This method was employed to generate 12 instances (N) all of which containing a different number of items to pack in varying strip widths.

The problem instance example used as an illustrative tool of the algorithms for Chapter 2, referred to as  $\mathcal{H}$ , is an instance of Burke, Kendall and Whitwell.

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

Hopper and Turton (2001) [42, 43] proposed three problem generators for creating guillotineable and non-guillotineable problem instances with known optimal solutions. The first follows the process of selecting a point at random in a large rectangle and

making vertical and horizontal cuts through the point so as to divide it into four parts and repeatedly iterates the process to generate the number of required items. The second generator bears similarity to the first with the exception that at each iteration only two parts are produced. These two generators create guillotineable problem instances. The third generator uses two points selected at random from a rectangle and then generates a pattern of five smaller rectangles in a non-guillotineable manner. The data set consists of three benchmark data sets labelled NT(n), NT(n), and C or CP. The first two consists of instances generated using the first and third problem generator algorithms respectively. The instances of C are classified into seven groups, each of three instances and each created by a different problem generator. The optimal packing heights are known for all instances. The data set can be obtained through the EURO Special Interest Group on Cutting and Packing (ESICUP) repository available at [30].

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

Wang and Valenzuela (2001) [91] proposed benchmark generators using recursive process principles which permitted restriction on the size (area ratio) and shape (aspect ratio) of the rectangles generated. Using their generator algorithms they introduced instances which fall into one of two sets, named the *pathological set* (path) and the *nice set*. Those instances belonging to the former set were constructed with no restrictions on the size and shape, while for the latter set size and shape constraints were imposed. The size constraint was specified so that the largest rectangle may not have an area larger than the area of the smallest rectangle by a factor of 7, and the shape constraint was imposed to ensure that the aspect ratio was confined to the interval  $[\frac{1}{4}, 4]$ . An interesting characteristic of this data set is that the item dimensions are all real numbers. All the problem sets have a strip width of 100 and the optimal packing height is known in every case.

### The Instances of Pinto and Oliveira (CX)

Pinto and Oliveira (2005) [71] created seven instances (CX) to study the relative performances of their proposed SPP algorithms. They did not specify the method of construction for these instances. The optimal solutions are known for each instance. These instances can be sourced from the online repository of Wei and Zhu at [96].

### The Instances of Bortfeldt and Gehring (AH)

Bortfeldt and Gehring (2006) [11] introduced a random number-based problem generator to create a variety of large benchmark instances (AH). The methodology employed the specification of four parameters of interest for the rectangles: a *maximum aspect ratio*, a *maximum area ratio*, a *heterogeneity ratio*, and a *width ratio*. The data set consists of a total 360 benchmark instances divided into 12 subsets with 30 instances each. The subsets differ in terms of the parameter values that were used to generate the problem instances contained in the subset. Optimal solutions are only known for some of the 360 instances. These instances can be sourced from the online repository of Wei and Zhu at [96].

### The Instances of Leung and Zhang (Zdf)

Leung and Zhang (2011) [52] proposed nine large problem instances (Zdf). These instances are amalgamations of a combination of existing zero-waste and non-zero waste instances. The instances belonging to the first class include those of Bengtsson [8] (beng) and Beasley [6] (gcut), and to the second include those of Burke *et al.* [15] (N) and Pinto and Oliveira [71] (CX). Of the nine, only five of the instances have known optimal solutions. These instances can be obtained through the EURO Special Interest Group on Cutting and Packing (ESICUP) repository available at [30].

---

## References

- [1] E. H. AARTS AND J. H. KORST, *Boltzmann machines for travelling salesman problems*, European Journal of Operational Research, 39 (1989), pp. 79–95.
- [2] R. ALVAREZ-VALDÉS, F. PARREÑO, AND J. M. TAMARIT, *Reactive grasp for the strip-packing problem*, Computers & Operations Research, 35 (2008), pp. 1065–1083.
- [3] J. E. ARONSON, T.-P. LIANG, AND R. V. MACCARTHY, *Decision support systems and intelligent systems*, vol. 4, Pearson Prentice-Hall Upper Saddle River, NJ, USA:, 2005.
- [4] A. R. BABU AND N. R. BABU, *Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms*, International Journal of Production Research, 37 (1999), pp. 1625–1643.
- [5] B. S. BAKER, E. G. COFFMAN, JR, AND R. L. RIVEST, *Orthogonal packings in two dimensions*, SIAM Journal on computing, 9 (1980), pp. 846–855.
- [6] J. BEASLEY, *An exact two-dimensional non-guillotine cutting tree search procedure*, Operations Research, 33 (1985), pp. 49–64.
- [7] G. BELOV AND H. ROHLING, *LP bounds in an interval-graph algorithm for orthogonal-packing feasibility*, Operations Research, 61 (2013), pp. 483–497.
- [8] B.-E. BENTGSSON, *Packing rectangular pieces—a heuristic approach*, The computer journal, 25 (1982), pp. 353–357.
- [9] J. O. BERKEY AND P. Y. WANG, *Two-dimensional finite bin-packing algorithms*, Journal of the operational research society, 38 (1987), pp. 423–429.
- [10] A. BORTFELDT, *A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces*, European Journal of Operational Research, 172 (2006), pp. 814–837.
- [11] A. BORTFELDT AND H. GEHRING, *New large benchmark instances for the two-dimensional strip packing problem with rectangular pieces*, in Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS’06), vol. 2, IEEE, 2006, pp. 30b–30b.
- [12] M. A. BOSCHETTI AND L. MONTALETTI, *An exact algorithm for the two-dimensional strip-packing problem*, Operations Research, 58 (2010), pp. 1774–1791.

- [13] A. BRINDLE, *Genetic algorithms for function optimization*, (1980).
- [14] E. BURKE, G. KENDALL, AND N. U. NOTTINGHAM, *Applying simulated annealing and the no fit polygon to the nesting problem*, in Proceedings of the world manufacturing congress, Citeseer, 1999.
- [15] E. K. BURKE, G. KENDALL, AND G. WHITWELL, *A new placement heuristic for the orthogonal stock-cutting problem*, Operations Research, 52 (2004), pp. 655–671.
- [16] E. K. BURKE, G. KENDALL, AND G. WHITWELL, *A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock-cutting problem*, INFORMS Journal on Computing, 21 (2009), pp. 505–516.
- [17] F. BUSETTI, *Simulated annealing overview*, 4 (2003). [Online], [Cited November 2020], Available from <http://www.aiinfinance.com/saweb.pdf>.
- [18] W. CHANG, J. CHENG, J. ALLAIRE, Y. XIE, J. MCPHERSON, ET AL., *Shiny: web application framework for R*, R package version, 1 (2017).
- [19] N. CHRISTOFIDES AND C. WHITLOCK, *An algorithm for two-dimensional cutting problems*, Operations Research, 25 (1977), pp. 30–44.
- [20] E. G. COFFMAN, JR, M. R. GAREY, D. S. JOHNSON, AND R. E. TARJAN, *Performance bounds for level-oriented two-dimensional packing algorithms*, SIAM Journal on Computing, 9 (1980), pp. 808–826.
- [21] J.-F. CÔTÉ, M. DELL’AMICO, AND M. IORI, *Combinatorial benders’ cuts for the strip packing problem*, Operations Research, 62 (2014), pp. 643–661.
- [22] CRAN, *CRAN: Available packages*, 2017. Available from <https://cran.r-project.org/>.
- [23] C. H. DAGLI AND P. POSHYANONDA, *New approaches to nesting rectangular patterns*, Journal of Intelligent Manufacturing, 8 (1997), pp. 177–190.
- [24] L. DAVIS, *Applying adaptive algorithms to epistatic domains.*, in IJCAI, vol. 85, 1985, pp. 162–164.
- [25] A. DE VRIES AND J. MEYS, *The Benefits of Using R*. [Online], [Cited November 2020], Available from <https://www.dummies.com/programming/r/the-benefits-of-using-r/>.
- [26] M. DELORME, *Mathematical models and decomposition algorithms for cutting and packing problems*, (2017).
- [27] M. DORIGO, M. BIRATTARI, AND T. STUTZLE, *Ant colony optimization*, IEEE computational intelligence magazine, 1 (2006), pp. 28–39.
- [28] H. DYCKHOFF, *A typology of cutting and packing problems*, European Journal of Operational Research, 44 (1990), pp. 145–159.



- [29] R. W. EGGLESE, *Simulated annealing: a tool for operational research*, European journal of operational research, 46 (1990), pp. 271–281.
- [30] ESICUP, *Datasets 2D-rectangular*, 2015. [Online], [Cited November 2020], Available from <https://www.euro-online.org/websites/esicup/data-sets/>.
- [31] Y. FANG AND J. LI, *A review of tournament selection in genetic programming*, in International Symposium on Intelligence Computation and Applications, Springer, 2010, pp. 181–192.
- [32] P. GILMORE AND R. E. GOMORY, *Multistage cutting stock problems of two and more dimensions*, Operations research, 13 (1965), pp. 94–120.
- [33] P. C. GILMORE AND R. E. GOMORY, *A linear programming approach to the cutting-stock problem*, Operations research, 9 (1961), pp. 849–859.
- [34] P. C. GILMORE AND R. E. GOMORY, *A linear programming approach to the cutting stock problem—part ii*, Operations research, 11 (1963), pp. 863–888.
- [35] F. GLOVER, *Future paths for integer programming and links to artificial intelligence*, Computers operations research, 13 (1986), pp. 533–549.
- [36] I. GOLAN, *Performance bounds for orthogonal oriented two-dimensional packing algorithms*, SIAM Journal on Computing, 10 (1981), pp. 571–582.
- [37] G. GÓMEZ-VILLOUTA, J.-P. HAMIEZ, AND J.-K. HAO, *Tabu search with consistent neighbourhood for strip packing*, in International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, Springer, 2010, pp. 1–10.
- [38] J.-P. HAMIEZ, J. ROBET, AND J.-K. HAO, *A tabu search algorithm with direct representation for strip packing*, in European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, 2009, pp. 61–72.
- [39] M. HIFI, *Exact algorithms for the guillotine strip cutting/packing problem*, Computers & Operations Research, 25 (1998), pp. 925–940.
- [40] M. HIFI, *The strip cutting/packing problem: incremental substrip algorithms-based heuristics*, Pesquisa Operacional, 19 (1999), pp. 169–188.
- [41] M. HIFI, *Library of instances*, 2004. [Online], [Cited November 2020], Available from <ftp://cermse.univ-paris1.fr/pub/CERMSEM/hifi/Strip-cutting/>.
- [42] E. HOPPER AND B. TURTON, *Problem generators for rectangular packing problems.*, Stud. Inform. Univ., 2 (2002), pp. 123–136.
- [43] E. HOPPER AND B. C. TURTON, *An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem*, European Journal of Operational Research, 128 (2001), pp. 34–57.
- [44] E. HOPPER AND B. C. TURTON, *A review of the application of meta-heuristic algorithms to 2d strip packing problems*, Artificial Intelligence Review, 16 (2001), pp. 257–300.

- [45] S. IMAHORI AND M. YAGIURA, *The best-fit heuristic for the rectangular strip packing problem: An efficient implementation and the worst-case approximation ratio*, Computers & Operations Research, 37 (2010), pp. 325–333.
- [46] S. JAKOBS, *On genetic algorithms for the packing of polygons*, European journal of operational research, 88 (1996), pp. 165–181.
- [47] A. JÚNIOR, *The Two-Dimensional Rectangular Strip Packing Problem*, PhD thesis, University of Porto, Porto, 2017.
- [48] M. KENMOCHI, T. IMAMICHI, K. NONOBE, M. YAGIURA, AND H. NAGAMOCCHI, *Exact algorithms for the two-dimensional strip packing problem with and without rotations*, European Journal of Operational Research, 198 (2009), pp. 73–83.
- [49] S. KIRKPATRICK, C. D. GELATT, AND M. P. VECCHI, *Optimization by simulated annealing*, science, 220 (1983), pp. 671–680.
- [50] B. KRÖGER, *Guillotineable bin packing: A genetic approach*, European Journal of Operational Research, 84 (1995), pp. 645–661.
- [51] K. LAI AND W. CHAN, *An evolutionary algorithm for the rectangular cutting stock problem*, International Journal of Industrial Engineering : Theory Applications and Practice, 4 (1997), pp. 130–139.
- [52] S. C. LEUNG AND D. ZHANG, *A fast layer-based heuristic for non-guillotine strip packing*, Expert Systems with Applications, 38 (2011), pp. 13032–13042.
- [53] S. C. LEUNG, D. ZHANG, AND K. M. SIM, *A two-stage intelligent search algorithm for the two-dimensional strip packing problem*, European Journal of Operational Research, 215 (2011), pp. 57–69.
- [54] T. LEUNG, C. YUNG, AND C. CHAN, *Applications of genetic algorithm and simulated annealing to the 2-dimensional non-guillotine cutting stock problem*, IFORS’99, (1999).
- [55] B. G. LINDNER, *Bi-objective generator maintenance scheduling for a national power utility*, PhD thesis, Stellenbosch: Stellenbosch University, 2017.
- [56] D. LIU AND H. TENG, *An improved bl-algorithm for genetic algorithm of the orthogonal packing of rectangles*, European Journal of Operational Research, 112 (1999), pp. 413–420.
- [57] A. LODI, S. MARTELLO, AND D. VIGO, *Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems*, INFORMS Journal on Computing, 11 (1999), pp. 345–357.
- [58] A. LODI, S. MARTELLO, AND D. VIGO, *Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem*, in Meta-Heuristics, Springer, 1999, pp. 125–139.
- [59] M. LOZANO, F. HERRERA, AND J. R. CANO, *Replacement strategies to maintain useful diversity in steady-state genetic algorithms*, in Soft Computing: Methodologies and Applications, Springer, 2005, pp. 85–96.

- [60] M. LUNDY AND A. MEES, *Convergence of an annealing algorithm*, Mathematical programming, 34 (1986), pp. 111–124.
- [61] S. MARTELLO, M. MONACI, AND D. VIGO, *An exact approach to the strip-packing problem*, INFORMS journal on Computing, 15 (2003), pp. 310–319.
- [62] S. MARTELLO AND D. VIGO, *Exact solution of the two-dimensional finite bin packing problem*, Management science, 44 (1998), pp. 388–399.
- [63] N. METROPOLIS, A. W. ROSENBLUTH, M. N. ROSENBLUTH, A. H. TELLER, AND E. TELLER, *Equation of state calculations by fast computing machines*, The journal of chemical physics, 21 (1953), pp. 1087–1092.
- [64] M. MITCHELL, *An introduction to genetic algorithms*, MIT press, 1998.
- [65] A. NECTOUX, *What is the way of packing oranges? — Kepler’s conjecture on the packing of spheres*, May 2015. [Online], [Cited October 2020], Available from <http://blog.kleinproject.org/?p=742>.
- [66] N. NTENE, *An algorithmic approach to the 2D oriented strip packing problem*, PhD thesis, Stellenbosch: Stellenbosch University, 2007.
- [67] N. NTENE AND J. H. VAN VUUREN, *A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem*, Discrete Optimization, 6 (2009), pp. 174–188.
- [68] F. G. ORTMANN, N. NTENE, AND J. H. VAN VUUREN, *New and improved level heuristics for the rectangular strip packing and variable-sized bin packing problems*, European Journal of Operational Research, 203 (2010), pp. 306–315.
- [69] F. G. ORTMANN AND J. H. VAN VUUREN, *Modified strip packing heuristics for the rectangular variable-sized bin packing problem*, ORiON, 26 (2010).
- [70] W. PANG, *DesktopDeployR*, 2020. [Online], Available from <https://github.com/wleepang/DesktopDeployR>.
- [71] E. PINTO AND J. F. OLIVEIRA, *Algorithm based on graphs for the non-guillotinable two-dimensional packing problem*, in Second ESICUP Meeting, Southampton, 2005.
- [72] D. POWER, *What is a dss*, The On-Line Executive Journal for Data-Intensive Decision Support, 1 (1997), pp. 223–232.
- [73] R CORE TEAM, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013. Available from <http://www.R-project.org/>.
- [74] R. G. RAKOTONIRAINY, *Metaheuristic solution of the two-dimensional strip packing problem*, PhD thesis, Stellenbosch University, Stellenbosch, 2018.
- [75] S. RAMESH, *Datasets 2D-rectangular*, February 2015. [Online], [Cited November 2020], Available from <https://blog.revolutionanalytics.com/2015/02/sharing-your-shiny-apps-1.html>.

- [76] S. RAMESH, *Sharing Your Shiny Apps*, February 2015. [Online], [Cited November 2020], Available from <https://blog.revolutionanalytics.com/2015/02/sharing-your-shiny-apps-1.html>.
- [77] K. RATANAPAN AND C. H. DAGLI, *An object-based evolutionary algorithm for solving rectangular piece nesting problems*, in 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, vol. 2, IEEE, 1997, pp. 989–994.
- [78] RSTUDIO TEAM, *Shiny*. [Online], [Cited November 2020], Available from <https://shiny.rstudio.com/>.
- [79] RSTUDIO TEAM, *Shiny – Sharing apps to run locally*, Jan 2014. [Online], [Cited November 2020], Available from <https://shiny.rstudio.com/articles/deployment-local.html>.
- [80] RSTUDIO TEAM, *RStudio: Integrated Development Environment for R*, RStudio, PBC., Boston, MA, 2020. Available from <http://www.rstudio.com/>.
- [81] C. SALTO, G. LEGUIZAMÓN, E. ALBA, AND J. M. MOLINA, *Hybrid ant colony system to solve a 2-dimensional strip packing problem*, in 2008 Eighth International Conference on Hybrid Intelligent Systems, IEEE, 2008, pp. 708–713.
- [82] C. SALTO, G. LEGUIZAMÓN, E. ALBA, AND J. M. MOLINA, *Evolutionary and ant colony optimization based approaches for a two-dimensional strip packing problem*, in Natural Intelligence for Scheduling, Planning and Packing Problems, Springer, 2009, pp. 245–266.
- [83] C. SIEVERT, *Interactive Web-Based Data Visualization with R, plotly, and shiny*, Chapman and Hall/CRC, 2020.
- [84] E. SILVA, J. F. OLIVEIRA, AND G. WÄSCHER, *2DCPackGen: a problem generator for two-dimensional rectangular cutting and packing problems*, European Journal of Operational Research, 237 (2014), pp. 846–856.
- [85] E. SILVA, J. F. OLIVEIRA, AND G. WÄSCHER, *2DCPackGen: A problem generator for two-dimensional rectangular cutting and packing problems*, European Journal of Operational Research, 237 (2014), pp. 846–856.
- [86] D. D. SLEATOR, *A 2.5 times optimal algorithm for packing in two dimensions*, Inf. Process. Lett., 10 (1980), pp. 37–40.
- [87] D. SMITH, *Bin packing with adaptive search*, in Proc. of the 1st International Conference on Genetic Algorithms and Their Applications, Pittsburgh PA, 1985, pp. 202–207.
- [88] P. J. VAN LAARHOVEN AND E. H. AARTS, *Simulated annealing*, in Simulated annealing: Theory and applications, Springer, 1987, pp. 7–15.
- [89] J. VAN VUUREN AND F. ORTMANN, *Benchmarks*, 2010. [Online], [Cited November 2020], Available from <https://www.vuuren.co.za/main.php>.

- [90] A. WALLACE AND L. WU, *packR*, 2020. [Online], Available from <https://github.com/ahwallace/2d-strip-packing-dss>.
- [91] P. Y. WANG AND C. L. VALENZELA, *Data set generation for rectangular placement problems*, European Journal of Operational Research, 134 (2001), pp. 378–391.
- [92] P. WARD, *The Crazy History of Tetris, Russia’s Most Famous Video Game*, Aug 2017. [Online], [Cited October 2020], Available from <https://theculturetrip.com/europe/russia/articles/the-crazy-history-of-tetris-russias-most-famous-video-game/>.
- [93] L. WEI, Q. HU, S. C. LEUNG, AND N. ZHANG, *An improved skyline based heuristic for the 2d strip packing problem and its efficient implementation*, Computers & Operations Research, 80 (2017), pp. 113–127.
- [94] L. WEI, W.-C. OON, W. ZHU, AND A. LIM, *A skyline heuristic for the 2D rectangular packing and strip packing problems*, European Journal of Operational Research, 215 (2011), pp. 337–346.
- [95] L. WEI, H. QIN, B. CHEANG, AND X. XU, *An efficient intelligent search algorithm for the two-dimensional rectangular strip packing problem*, International Transactions in Operational Research, 23 (2016), pp. 65–92.
- [96] L. WEI AND W. ZHU, *Skyline Heuristic for the 2D Rectangular Packing and Strip Packing Problems*, 2011. [Online], [Cited November 2020], Available from <https://www.computational-logistics.org/orlib/topic/2D%20Strip%20Packing/index.html>.
- [97] L. WHITNEY, *R programming language continues to grow in popularity*, September 2020. [Online], [Cited November 2020], Available from <https://www.techrepublic.com/article/r-programming-language-continues-to-grow-in-popularity/>.
- [98] S. YANG, S. HAN, AND W. YE, *A simple randomized algorithm for two-dimensional strip packing*, Computers & operations research, 40 (2013), pp. 1–8.
- [99] X.-S. YANG, *Nature-inspired optimization algorithms*, Elsevier, 2014.
- [100] L. H. YEUNG AND W. K. TANG, *Strip-packing using hybrid genetic approach*, Engineering Applications of Artificial Intelligence, 17 (2004), pp. 169–177.