Exact and Evolutionary Algorithms for the Score-Constrained Packing Problem

Asyl Liyakat Hawa

School of Mathematics
Cardiff University

A thesis submitted for the degree of
Doctor of Philosophy

September 2020

Summary

This thesis concerns the Score-Constrained Packing Problem (SCPP), a combinatorial optimisation problem related to the one-dimensional bin packing problem. The aim of the SCPP is to pack a set of rectangular items from left to right into the fewest number of bins such that no bin is overfilled; however, the order and orientation of the items in each bin affects the feasibility of the overall solution. The SCPP has applications in the packaging industry, and obtaining high quality solutions for instances of the SCPP has the ability to reduce the amount of waste material, costs, and time, which motivates the study in this thesis.

The minimal existing research on the SCPP leads us to explore a wide range of approaches to the problem in this thesis, implementing ideas from related problems in literature as well as bespoke methods. To begin, we present an exact algorithm that can produce a feasible configuration of a subset of items in a single bin in polynomial-time. We then introduce a range of methods for the SCPP including heuristics, an evolutionary algorithm framework comprising a local search procedure and a choice of three distinct recombination operators, and two algorithms combining metaheuristics with an exact procedure. Each method is investigated to gain more insight into the characteristics that benefit or hinder the improvement of solutions, both theoretically and computationally, using a large number of problem instances with varying parameters. This allows us to determine the specific methods and properties that produce superior solutions depending on the type of problem instance.

Declarations

This work has not been submitted in substance for any other degree or award at this or any other university or place of learning, nor is it being submitted concurrently in candidature for any degree or other award.

Signed ..... (candidate) Date .....

Statement 1

This thesis is being submitted in partial fulfillment of the requirements for the degree of PhD.

Signed ..... (candidate) Date .....

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated, and the thesis has not been edited by a third party beyond what is permitted by Cardiff University's Policy on the Use of Third Party Editors by Research Degree Students. Other sources are acknowledged by explicit references. The views expressed are my own.

Signed ..... (candidate) Date .....

Statement 3

I hereby give consent for my thesis, if accepted, to be available online in the University's Open Access repository and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate) Date .....

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Acknowledgements

Firstly, I would like to thank my supervisors, Rhyd and Jonathan, for their unwavering support over the past four years. Thank you both for your words of encouragement, your patience, and for allowing me to pursue my own ideas.

I would also like to extend my gratitude to EPSRC for their financial support and the School of Mathematics for providing me with the resources required for this research. In addition, to all the members of staff and my fellow PhD students at Cardiff University School of Mathematics, thank you for making this an enjoyable experience full of wonderful memories.

Finally, I would like to thank my family. Mum and Dad, thank you for all of the sacrifices you have made over the past 30 years for your children. I don't know if I would have had the opportunity to go to university let alone do a PhD without you. Thank you for always listening and for your unconditional love. Isra, thank you for taking me away when I've needed a break and for keeping me grounded, and Hamza, thank you for your ridiculous jokes and for being there when I've needed someone to talk to about work. To my nephew Faris, thank you for always putting a smile on my face when I'm with you. I love you all very much.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

From sorting belongings into boxes when moving house to loading goods into delivery vans, packing problems occur frequently – yet often unknowingly – in our everyday lives. Naturally, in order to reduce costs and prevent waste, the aim of the majority of such problems is to pack all of the "items" efficiently into the "bins" so as to make use of all of the space provided, whilst also attempting to minimise the number of bins required. In some cases this also has an environmental benefit, reducing vehicle emissions and the demand for materials. The conditions for each packing problem can vary. For example, when moving house it is often desirable to pack items from the same room or category into the same set of boxes, whereas supermarkets may choose to organise goods into vehicles in the order they are to be delivered to customers. In this thesis, we focus on a specific packing problem with conditions involving the order and orientation of the items in the bins: the Score-Constrained Packing Problem (SCPP).

1.1 Definitions

The main problem we address in this thesis is based on an issue that first emerged in the packaging industry, where boxes are to be produced from strips of corrugated cardboard. The first stage of production involves cutting flat sheets from the cardboard so as to minimise waste material, and has been widely researched as the cutting stock problem (see Chapter 2). The second stage consists in a combinatorial problem presented by Goulimis in 2004, and requires scoring the flat sheets in specific locations to make it easier to fold the cardboard into boxes.

Consider a set $\mathcal{I}$ of n rectangular items of fixed height H, where each item $i \in \mathcal{I}$ has width $w_i \in \mathbb{Z}^+$. In addition, each item is marked with two vertical score lines in predetermined places. The distance between each score line and the nearest edge of the item are the score widths, $a_i, b_i \in \mathbb{Z}^+$ (where without loss of generality $a_i \leq b_i$).

An example of an item i with these dimensions is provided in Figure 1.1.

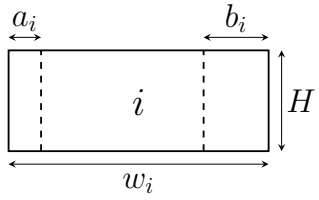


Figure 1.1: The dimensions of an item i with height H, width w_i, and score widths a_i and b_i. The dashed lines on the item indicate the vertical score lines.

The machine used to score the items consists of pairs of knives secured to a bar. Each pair of knives simultaneously cut along the score lines of two adjacent items, which allows the cardboard to be folded with ease at a later stage. However, due to the manner in which the machine is designed the knives in each pair must maintain a set distance from one another, a so-called "minimum scoring distance" $\tau \in \mathbb{Z}^+$, which is approximately 70mm in industry. Therefore, in order for the knives to score the items in the correct locations, the distance between the score lines of adjacent items must be equal to or exceed the minimum scoring distance. Hence, the following *vicinal sum constraint* must be fulfilled:

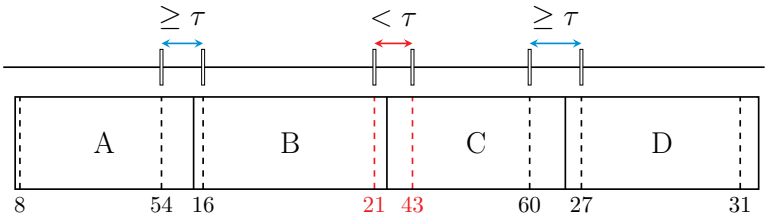
$$\mathbf{r}(i) + \mathbf{l}(i+1) \geq \tau \quad \forall\, i \in \{1, 2, \ldots, n-1\}, \tag{1.1}$$

where $\mathbf{l}(i)$ and $\mathbf{r}(i)$ denote the left- and right-hand score widths of the ith item in the alignment. Each item $i \in \mathcal{I}$ can be placed in one of two orientations: *regular*, denoted (a_i, b_i), where the smaller score width a_i is on the left-hand side of item i, or *rotated*, denoted (b_i, a_i), where the larger score width b_i is on the left-hand side. Note that the outermost score widths in the alignment are not included in the above constraint as they are not adjacent to any other items, and the outermost score lines are cut individually. Obviously, if the vicinal sum constraint is satisfied the distance between the adjacent score lines will be sufficient for the knives to be able to operate correctly.

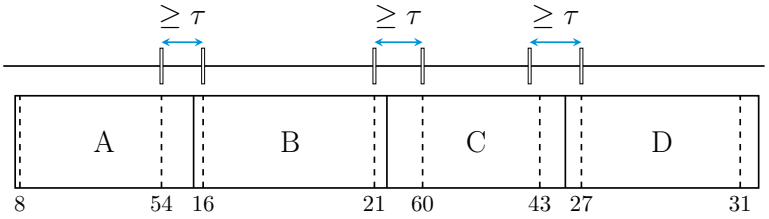
Figure 1.2 shows how adjacent items are scored simultaneously by pairs of knives mounted on a bar. In Figure 1.2a, each item has been placed in a regular orientation, and although the vicinal sum constraint is satisfied between items A and B and items C and D the full arrangement of all four items is infeasible as the sum of the adjacent score widths of items B and C is less than the minimum scoring distance τ; thus the knives are unable to move close enough together to score the items along the marked lines. A feasible alignment of the four items can be obtained by placing item C in a rotated orientation, as shown in Figure 1.2b.

In practice, if an arrangement violates the vicinal sum constraint a "double run" process can be utilised. This process involves scoring the items individually at a much

slower speed, incurring additional costs. Therefore, in this work, any arrangement of items in which the vicinal sum constraint is not fulfilled is considered infeasible.



(a) A infeasible alignment of four items which have been placed in regular orientations



(b) A feasible alignment of four items obtained by placing item C in a rotated orientation

Figure 1.2: Examples of infeasible and feasible arrangements of four items to be scored by pairs of knives. (a) The red score lines show where the vicinal sum constraint (1.1) is violated between items B and C; and (b) rotating item C results in a feasible arrangement. Here, $\tau = 70$.

There are $2^{n-1}n!$ distinct configurations of n items excluding symmetry, and so for large n a naive algorithm relying on complete enumeration can quickly lead to a combinatorial explosion – even for just 10 items, there are almost two billion arrangements. Consequently, other methods are required to find feasible orderings within a reasonable amount of time.

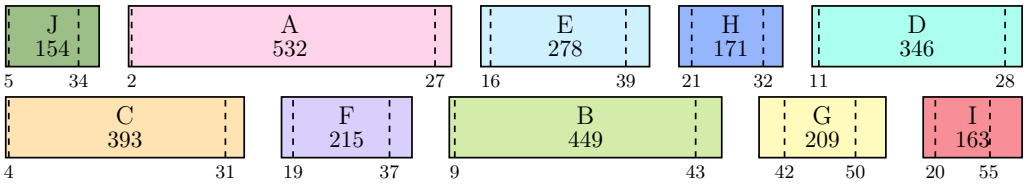
Recall that the items are to be cut and scored from flat sheets of cardboard. As we have seen in Goulimis' proposed problem above, the items' widths are disregarded as the aim is to simply find a feasible arrangement of the given items onto a single strip of cardboard of seemingly infinite width. However, in industrial application sheets of material are often provided in fixed, finite widths. Thus, given large problem instances, multiple strips may be required to accommodate all of the items.

Note that the problems studied in this thesis can be classified as variants of the bin packing problem (see Chapter 2) according to Wäscher et al. (2007); thus for simplicity in the remainder of this thesis we will refer to items being "packed" into "bins" as opposed to being "cut" from "strips".

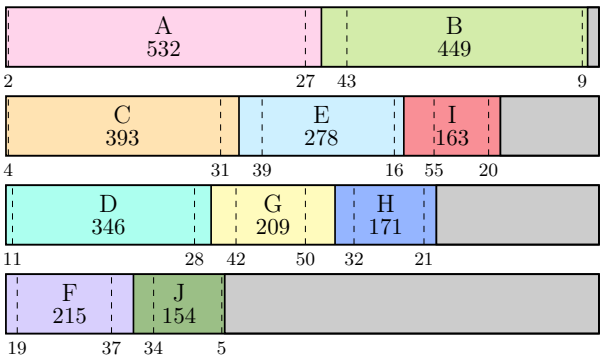
Let us now formally define the main problem of this thesis:

Definition 1.1 *Let $\mathcal{I}$ be a set of n rectangular items of height $H > 0$ with varying widths $w_i \in \mathbb{Z}^+$ and score widths $a_i, b_i \in \mathbb{Z}^+$ for all $i \in \mathcal{I}$. Given a minimum scoring distance $\tau \in \mathbb{Z}^+$, the* Score-Constrained Packing Problem (SCPP) *involves packing the items from left to right into the fewest number of $H \times W$ bins such that the vicinal sum constraint is satisifed in each bin and no bin is overfilled.*

Figure 1.3 shows an instance $\mathcal{I}$ of the SCPP consisting of 10 items and a corresponding feasible solution, where four bins are required to accommodate the items whilst fulfilling the vicinal sum constraint in each bin.



(a)



(b)

Figure 1.3: (a) An example instance $\mathcal{I}$ of the SCPP comprising 10 items; and (b) a feasible solution for the instance using four bins of capacity $W = 1000$. The remaining space in each bin in shaded in grey. Here, $\tau = 70$.

For the SCPP, the set $\mathcal{F}$ (defined in Chapter 2) is the set of all item subsets that can be feasibly packed into a single bin such that the vicinal sum constraint is fulfilled. Therefore, the SCPP not only involves deciding which items should be packed into which bin, but also *how* the items should be packed – that is, determining the order and orientation of items within each bin. Thus, there exists a sub-packing problem within each bin, defined as follows:

Definition 1.2 *Let $\mathcal{I}' \subseteq \mathcal{I}$ be a subset of rectangular items whose total width is less than or equal to the bin capacity, i.e. $A(\mathcal{I}') = \sum_{i \in \mathcal{I}'} w_i \leq W$. Then, given a minimum scoring distance $\tau \in \mathbb{Z}^+$, the* Score-Constrained Packing Sub-Problem (sub-SCPP) *consists in finding an ordering and orientation of the items in $\mathcal{I}'$ such that the vicinal sum constraint is satisfied.*

1.2 Research Aims

The main aims of the research in this thesis are as follows:

1. To compare several heuristics for the SCPP and identify characteristics that make the heuristics suitable for particular problem instance types.

2. To investigate the effects of different recombination operators within an evolutionary algorithm framework on the quality of solutions and determine the desirable attributes of each operator.

3. To explore the combination of exact methods with metaheuristics and assess the associated advantages and disadvantages.

4. To determine whether improved solutions can be found by relaxing elements of the SCPP, namely by allowing spaces between items in each bin.

1.3 Contributions of the Thesis

The contributions of this thesis are as follows:

- In Chapter 3, we present the Alternating Hamiltonian Construction (AHC) algorithm, an exact polynomial-time algorithm for solving the sub-SCPP. This algorithm is an improved, extended version of a procedure proposed by Becker (2010), however this previous procedure merely provided a simple statement of feasibility or infeasibility. Our AHC algorithm not only determines the existence of a solution for any given instance of the sub-SCPP, but also has the ability to provide the order and orientation of the items in a feasible solution, which is extremely beneficial in industrial applications. Furthermore, we also highlighted and rectified an issue within the previous procedure which produced invalid solutions to feasible problem instances.

- In Chapter 4, three novel heuristics developed for the SCPP are introduced which, at the time of writing, are the only known heuristics for the problem. Two of the heuristics, MFFD and MFFD$^+$, are based on the well-known FFD heuristics for the bin packing problem, where MFFD$^+$ incorporates the AHC algorithm from the previous chapter, whilst the final heuristic, PS, focuses on packing individual bins in turn. Experimental results show that PS and MFFD$^+$ produce superior solutions depending on the proportions of score widths in the given problem instance that meet the vicinal sum constraint.

- In Chapter 5, we describe an evolutionary algorithm (EA) framework comprising three custom recombination operators for the SCPP which ensure solution feasibility, as well as a local search procedure to improve solution quality in each iteration. By comparing the performance of the distinct recombination operators, we are able to determine the two most desirable features of the operators that produce high quality solutions: collecting high quality groups of bins

to propagate throughout the whole population, and prioritising the selection of bins containing a larger number of items.

- In Chapter 6, the Minimum Cardinality Exact Cover Problem (MXCP) is defined and its relation to the SCPP is explored. From this, we design two algorithms for the SCPP combining an exact solver for the MXCP with different evolutionary operators from our evolutionary algorithm framework in the previous chapter. Experimental results show the advantages of obtaining and utilising feedback from solutions for subsequent iterations, a characteristic not present in our EA, and the disadvantages of implementing an exact solver within a procedure.

- In Chapter 7, we present the Modified Score-Constrained Packing Problem (MSCPP) and associated sub-problem, an novel alternative version of the SCPP in which "gaps" are permitted between adjacent items in bins in order to fulfil the vicinal sum constraint. An adaptation of the AHC algorithm is proposed for the sub-problem, along with an MFFD$^+$-based heuristic for the MSCPP. A comparison of the two problems shows that although solutions to the MSCPP will clearly comprise fewer bins, it may not be possible to develop an exact polynomial-time algorithm for the sub-problem as with the AHC algorithm for the sub-SCPP.

1.4 Outline of the Thesis

The thesis is dedicated to the development of algorithms and heuristics for the SCPP. The next chapter provides an overview into existing literature on packing and grouping problems and describes the unique properties of the SCPP. Chapter 3 focuses on the sub-SCPP, first discussing the possible ways of modelling the problem before detailing an exact polynomial-time algorithm for solving the sub-SCPP. In Chapter 4 we begin our approach to the SCPP and compare the performance of heuristics developed specifically for the SCPP. An evolutionary approach to the SCPP is presented in Chapter 5, which includes a local search procedure and a comparison of three distinct recombination operators. Chapter 6 introduces two algorithms combining metaheuristics and an exact procedure involving a recursive backtracking algorithm, and provides an ILP formulation for the SCPP. In Chapter 7 we briefly introduce an alternative version of the SCPP and examine the differences in solutions for each problem. Finally, Chapter 8 gives concluding remarks and discusses outcomes and possible directions for further work.

1.5 Academic Publications

The following academic articles have been produced from the research in this thesis:

- Hawa, A.L., Lewis, R. and Thompson, J.M. (2018). Heuristics for the Score-Constrained Strip-Packing Problem. In *International Conference on Combinatorial Optimization and Applications* (pages 449–462). Springer.
- Hawa, A.L., Lewis, R. and Thompson, J.M. (2020). Exact and Approximate Methods for the Score-Constrained Packing Problem. *Under second review with the European Journal of Operational Research.*

Chapter 2

Literature Review

In this chapter, we start by providing a brief overview to computational complexity, which allows us to assess the difficulty of various combinatorial optimisation problems. We then introduce the one-dimensional bin packing problem (BPP) and discuss the methods that have been used to solve the problem by reviewing relevant literature, before exploring alternative packing problems related to the BPP. The Score-Constrained Packing Problem (SCPP) is also examined and compared to the various packing problems highlighted throughout the chapter.

2.1 Computational Complexity

To begin, let us distinguish between two types of problem.

Definition 2.1 *A* decision problem *is a problem in which the answer is either "yes" or "no", depending on the input. An* optimisation problem *is a problem where an optimal solution is to be found from all feasible solutions, with respect to some given objective function.*

Note that an optimisation problem can in some cases be infeasible - that is, no feasible solution exists for the given problem. Observe that optimisation problems can be transformed into decision problems and vice versa. Consider, for example, the travelling salesman problem (TSP). The optimisation variant of the problem is: "find the tour with the shortest length", whilst the decision variant is: "for each value D, does there exist a tour with length less than D?". It can be seen that if the decision problem is asked and answered repeatedly, the length of the shortest tour can be determined.

The field of computational complexity theory concentrates on classifying problems into distinct complexity classes, which allows us to compare the difficulty of different problems. The classification of a problem is determined by the number

of machine operations, or running time, needed by an algorithm to find or verify a solution to the problem relative to the input size.

An algorithm is said to be of *polynomial-time* if its running time is upper-bounded by a polynomial expression in terms of the input size for the algorithm. Such algorithm are considered to be efficient algorithms (Edmonds, 1965; Cobham, 1965). Consequently, a problem is said to be *polynomial* if there exists a polynomial-time algorithm for finding a solution for any instance of the problem. The complexity class P contains all polynomial decision problems. One other way of describing P is that the decision problems in P can be solved by a *deterministic* Turing machine.

There are many problems for which no polynomial-time algorithms for finding solutions have yet been found. Unfortunately, it has also not been determined that there do not exist polynomial-time algorithms for such problems. For some of these problems, however, it is possible to *verify* a solution in a reasonable amount of time. A problem is said to be *non-deterministic polynomial* if there exists a polynomial-time algorithm that is able to verify any given solution to the problem (Cook, 1971). The complexity class NP contains all non-deterministic polynomial decision problems. Again, NP can be described as the set of decision problems for which a given solution can be verified in polynomial-time by a deterministic Turing machine, or that is solvable by a *non-deterministic* Turing machine in polynomial-time. Clearly, as non-deterministic Turing machines are theoretical, it is thought that problems in NP will always be intractable to solve with our regular (deterministic) machines; thus, is it widely assumed that P$\neq$NP.

A problem is said to be NP-hard if every problem in NP can be reduced by a polynomial-time algorithm into that problem. Note, however, that NP-hard problems do not have to be in NP and may not even be decision problems – they can also be optimisation or search problems. On the other hand, a problem is said to be NP-complete if it is NP-hard *and* is an element of NP. NP-complete problems are considered to be the hardest problems in NP, whilst NP-hard problems are at least as hard as the problems in NP-complete. Therefore, if there exists a polynomial-time algorithm that can *solve* an NP-hard problem it would then be possible to solve all problems in NP in polynomial-time, and so P=NP. Figure 2.1 illustrates the relationship between these complexity classes assuming P$\neq$NP.

Many surveys, articles, and books covering computational complexity theory are readily available, including overviews on the topic (Cook, 1983), pioneering work on NP-completeness (Karp, 1972), discussions into the importance of considering the equality of P and NP (Cook, 2006), as well as general books exploring various aspects of the field (Garey and Johnson, 1979; Papadimitriou, 2003). In addition, further information on Turing machines can be found in the notable works of Petzold (2008)

and Jongen et al. (2007).

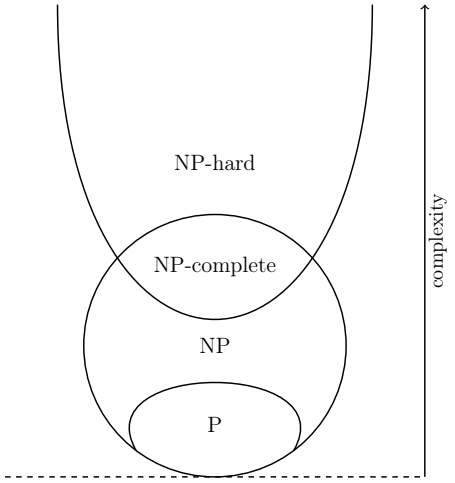


Figure 2.1: A diagram showing the relationship between the complexity classes P, NP, NP-hard, and NP-complete under the assumption that P≠NP.

2.2 Grouping Problems

Many combinatorial optimisation problems involve the grouping or partitioning of elements into subsets. These types of problems can be seen in areas such as scheduling (Thompson and Dowsland, 1998; Carter et al., 1996), frequency assignment (Aardal et al., 2007), graph colouring (Lewis et al., 2012; Malaguti et al., 2008), and load balancing (Rekiek et al., 1999), as well as in practical problems in computer science such as table formatting, prepaging, and file allocation (Garey et al., 1972). Formally, given a set $\mathcal{I}$ of n elements, the aim is to produce a set of groups $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ such that:

$$\bigcup_{j=1}^{k} S_j = \mathcal{I}, \tag{2.1a}$$

$$S_i \cap S_j = \emptyset \quad i, j = 1, 2, \ldots, k,\ i \neq j, \tag{2.1b}$$

$$S_j \in \mathcal{F} \quad j = 1, 2, \ldots, k. \tag{2.1c}$$

Conditions (2.1a) and (2.1b) enforce the requirement that every element in $\mathcal{I}$ must be in exactly one of the k groups in $\mathcal{S}$, whilst Condition (2.1c) specifies that each group $S_j \in \mathcal{S}$ must be *feasible*. Here, we introduce the set $\mathcal{F}$, which denotes the set of all feasible subsets of elements in $\mathcal{I}$. The notion of feasibility is dependent on the particular constraints of the given problem.

The graph colouring problem (GCP) involves assigning colours to vertices on a graph G such that no two adjacent vertices are of the same colours, with the objective of minimising the total number of colours used. The chromatic number of a graph

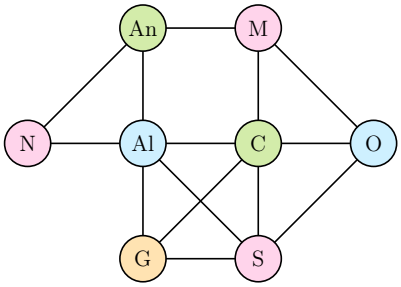
G, denoted $\chi(G)$, is the minimum number of colours required to colour G. The GCP is one example of a grouping problem, where the vertices of the graph must be grouped into the fewest non-overlapping subsets (i.e. colour classes) such that each subset is an independent set. For this problem, $\mathcal{F}$ contains all possible independent sets of vertices on the graph. Figure 2.2b illustrates an example solution of the GCP on a graph G comprising eight vertices, where $\chi(G) = 4$ colours have been used to colour the vertices feasibly. For a deeper introduction to graph colouring, the reader is directed to the work of Lewis (2015b).

Another example of a grouping problem occurs when scheduling exams in schools and universities. In its simplest form, the problem involves sorting exams into a limited number of timeslots whereby no student is scheduled to sit more than one exam per timeslot. In practice, however, many other factors must be considered such as the exam locations, the capacity of each location, and restrictions on the order of specific exams. For this problem, each timeslot can be considered as a group; thus $\mathcal{F}$ contains all possible groups of exams where each student is attending at most one exam.

It is interesting to note that the exam scheduling problem can in fact be reformulated as a graph colouring problem. Suppose each exam i is represented by a vertex v_i on a graph G, and for each pair of exams i and j there exists an edge $\{v_i, v_j\}$ on G if at least one student is required to sit both exams. Then, the resulting graph $G = (V, E)$ depicts the exams than can and cannot be scheduled into the same timeslot. By finding a solution to the GCP with respect to G, the fewest number of timeslots for which all exams can be scheduled without clashes can be determined. Figure 2.2a shows the number of students required to sit each pair of exams for the subjects **Al**gebra, **An**alysis, **C**alculus, **G**eometry, **M**echanics, **N**umber Theory, **O**ptimisation, and **S**tatistics. The colours of the vertices on the graph G, shown in Figure 2.2b, indicate the exams that should be scheduled into the same timeslots.

	Al	An	C	G	M	N	O	S
Al	–	12	35	26	0	44	0	19
An	12	–	0	0	31	40	0	0
C	35	0	–	52	29	0	16	37
G	26	0	52	–	0	0	0	46
M	0	31	29	0	–	0	54	0
N	44	40	0	0	0	–	0	0
O	0	0	16	0	54	0	–	38
S	19	0	37	46	0	0	38	–

(a)



(b)

Figure 2.2: (a) An example instance of the exam scheduling problem, where the table shows for each pair of subjects the number of students sitting both exams; and (b) the graph G depicting the scheduling problem. Solving the GCP on G shows that four timeslots are required, with the vertex colours indicating the exams that should be assigned to the same timeslot.

A grouping problem will often have one of two objectives regarding the number of groups k in a solution $\mathcal{S}$. One objective involves determining whether a solution $\mathcal{S}$ exists that contains exactly k groups, where k has been explicity specified. Grouping problems with this type of objective are decision problems. The other objective is to find a solution comprising the fewest number of groups k; these types of grouping problems are optimisation problems. This latter class of problems, identified by Lewis (2009) as Minimum Grouping Problems (MGPs), can be separated into two sub-classes: Order Dependent Minimum Grouping Problems (ODMGPs), where the order of the groups affects the feasibility and/or quality of the solution – for example, the exam scheduling problem with an additional condition preventing students from sitting exams in consecutive timeslots – and Order Independent Minimum Grouping Problems (OIMGPs), in which the order of the groups within a solution is irrelevant.

2.3 The One-Dimensional Bin Packing Problem

One particular class of grouping problem with applications in the material and construction industries are cutting and packing problems. These problems involve cutting items from stock material or packing items into bins subject to specific constraints. Perhaps the most well-known of these problems is the one-dimensional bin packing problem (BPP).

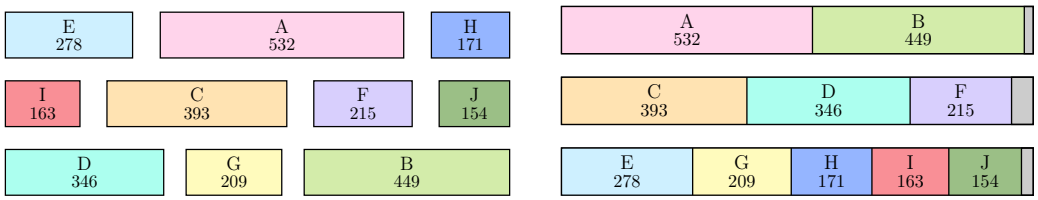
Definition 2.2 *Given a set $\mathcal{I}$ of n rectangular items of varying widths $w_i \in \mathbb{Z}^+$ and equal height $H > 0$ for all items $i \in \mathcal{I}$, the* one-dimensional bin packing problem (BPP) *involves packing all items in $\mathcal{I}$ into the fewest number of $H \times W$ bins such that no bin is overfilled.*

The one-dimensional bin packing problem is a classical combinatorial optimisation problem originating back to the 1930s (Kantorovich, 1960). The problem requires the items in $\mathcal{I}$ to be grouped into subsets $\mathcal{S} = \{S_1, \ldots, S_k\}$ of bins with the aim of minimising the number of bins k. For this problem, a bin S_j is feasible if the total widths of all the items in S_j does not exceed the bin capacity W; that is, $A(S_j) = \sum_{i \in S_j} w_i \leq W$. A basic lower bound for the fewest number of bins k in a solution is the theoretical minimum, which can be calculated in $O(n)$ time (Martello and Toth, 1990b):

$$t = \left\lceil \frac{\sum_{i=1}^{n} w_i}{W} \right\rceil. \tag{2.2}$$

Figure 2.3 depicts an example instance $\mathcal{I}$ of 10 items for the BPP to be packed into bins of capacity $W = 1000$ along with a feasible solution comprising three bins. As

the theoretical minimum for this problem instance is $t = 3$, the solution is in fact optimal.



(a) $\mathcal{I}$ (b) Solution for $\mathcal{I}$

Figure 2.3: (a) An example instance $\mathcal{I}$ of the BPP comprising 10 items; and (b) a feasible solution for the instance using three bins of capacity $W = 1000$. The remaining space in each bin is shaded in grey. As the theoretical minimum $t = 3$, the solution is optimal.

Derived from the one-dimensional bin packing problem are the two-dimensional (Lodi et al., 2002) and three-dimensional (Martello et al., 2000) bin packing problems. The BPP has a variety of real world applications, such as assigning advertisements to programme breaks in television broadcasting (Brown, 1971), allocating memory in computer network designs (Chandra et al., 1978), scheduling tasks on multiple processors (Coffman et al., 1978; van de Vel and Shijie, 1991), and packing items into containers with size or weight capacity constraints (Eilon and Christofides, 1971; Hung and Brown, 1978).

The BPP can be formulated as the following integer linear program:

$$\text{minimise} \quad \sum_{j=1}^{n} y_j \tag{2.3a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_{ij} \leq W y_j \qquad j = 1, \ldots, n \tag{2.3b}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{2.3c}$$

$$y_j \in \{0, 1\} \qquad j = 1, \ldots, n \tag{2.3d}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n \tag{2.3e}$$

$$y_j = \begin{cases} 1 & \text{if bin } j \text{ is used} \\ 0 & \text{otherwise} \end{cases} \qquad x_{ij} = \begin{cases} 1 & \text{if item } i \text{ is packed into bin } j \\ 0 & \text{otherwise.} \end{cases}$$

The objective 2.3a is to minimise the number of bins of capacity W used to pack the items. Constraint 2.3b ensures that the bins used in the solution are not overfilled, whilst Constraint 2.3c states that each item must be packed into exactly one of the bins in the solution. The final two constraints restrict the variables x_{ij} and y_j to the integers 0 and 1.

2.3.1 Computational Complexity of the Bin Packing Problem

The partition problem asks the question: "given a set S of positive integers, can S be partitioned into two subsets S_1 and S_2 such that the sum of the elements in S_1 is equal to the sum of the elements in S_2?" This decision problem is known to be NP-complete, and is in fact one of Karp's original 21 NP-complete problems (1972). As discussed in Section 2.1, by reducing the partition problem to the BPP we are able to show that the BPP is also NP-complete.

Theorem 2.3 *Given an instance $\mathcal{I}$ of the BPP, the problem of deciding whether there exists a solution to $\mathcal{I}$ comprising two bins is NP-complete.*

Proof. We reduce the partition problem, which we know to be NP-complete, to the above bin packing problem. Given an instance $S = \{s_1, \ldots, s_n\}$ of the partition problem, consider the instance $\mathcal{I} = \{w_1, \ldots, w_n\}$ of the BPP, where

$$w_i = \frac{2s_i}{\sum_{j=1}^{n} s_j} \quad \forall\, i = 1, \ldots, n.$$

Clearly, there exists a solution for the BPP using two bins if and only if there exists a subset $S_1 \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S_1} s_i = \sum_{i \notin S_1} s_i$. □

As the decision variant of the BPP is NP-complete, it follows that the optimisation variant of the BPP, provided in Definition 2.2, is NP-hard (Garey and Johnson, 1978, 1979).

2.4 Heuristics for the Bin Packing Problem

As the BPP is NP-hard, under the generally accepted assumption that P≠NP we cannot hope to find an optimal solution in polynomial-time for all instances of the BPP. Consequently, much research has been devoted to developing algorithms that run in polynomial-time that produce near-optimal solutions, such as heuristics and approximation algorithms. Heuristics aim to produce feasible solutions by trading optimality, accuracy, or precision for speed; thus solutions created by heuristics are not guaranteed to be optimal. Approximation algorithms are heuristics that have the advantage of producing solutions that are provably close to the optimal solution, which makes them much more desirable in theory. It is important to note, however, that the worst-case bounds obtained from approximation algorithms cover all instances of a problem including unusual instances that would not appear in practice, and so it tends to be that heuristics derived from approximation algorithms produce solutions that are much closer to the optimal than suggested by the bounds of the approximation algorithm (Williamson and Shmoys, 2011).

Let $\mathcal{A}(\mathcal{I})$ denote the number of bins in a solution for an instance $\mathcal{I}$ produced using an approximation algorithm $\mathcal{A}$, and let $\text{OPT}(\mathcal{I})$ denote the number of bins in the optimal solution for the instance $\mathcal{I}$. Then, $\mathcal{A}$ has an approximation ratio $c > 1$ if for any instance $\mathcal{I}$, $\mathcal{A}(\mathcal{I}) \leq c\text{OPT}(\mathcal{I}) + b$, where b is a positive constant. Clearly, the goal is to obtain an approximation ratio c that is as close to 1 as possible. However, Vazirani (2003) has shown that there cannot exist an approximation algorithm with $c < \frac{3}{2}$ for the BPP unless P=NP.

One of the most well-known heuristics for the BPP is First-Fit (FF), a greedy online algorithm that packs each item, given in some arbitrary order, into the lowest-indexed bin that can feasibly accommodate the item and opening a new bin when required, as shown in Algorithm 1. It is known that there always exists at least one ordering of the items such that FF produces an optimal solution (Lewis, 2009).

Algorithm 1 $\text{FF}(\mathcal{I})$

```
A(S_j) ← 0 ∀ j = 1, ..., |I|          ▷ All bins are initially empty
S ← ∅
for i ← 1 to |I| do
    j ← 1
    packed ← false
    while not packed do
        if S_j ∈ S then
            if A(S_j) + w_i ≤ W then
                S_j ← S_j ∪ {i}
                A(S_j) ← A(S_j) + w_i
                packed ← true
            else j ← j + 1
        else if S_j ∉ S then
            S_j ← S_j ∪ {i}
            A(S_j) ← A(S_j) + w_i
            S ← S ∪ {S_j}
            packed ← true
return S
```

It was first shown by Ullman (1971) that for any instance $\mathcal{I}$, $\text{FF}(\mathcal{I}) \leq \frac{17}{10}\text{OPT}(\mathcal{I}) + 3$. This was then improved twice by Garey et al. (1972, 1976), before Xia and Tan presented the upper bound of $\text{FF}(\mathcal{I}) \leq \frac{17}{10}\text{OPT}(\mathcal{I}) + \frac{7}{10}$ in 2010. The most recent bound was proven by Dósa and Sgall (2013) to be exactly $\frac{17}{10}\text{OPT}(\mathcal{I})$, concluding that the bound is tight. A number of related heuristics were presented in Johnson's seminal thesis in 1973 including the Best-Fit (BF) heuristic, which operates by packing each item into the bin with the least amount of remaining space. BF has been shown to have similar bounds to the FF heuristic (Dósa and Sgall, 2014).

An enhancement on the FF and BF heuristics yields their offline counterparts, First-Fit Decreasing (FFD) and Best-Fit Decreasing (BFD), where the items are

packed in non-increasing order of size (Johnson et al., 1974). In his thesis, Johnson (1973) provided the initial bound $\text{FFD}(\mathcal{I}) \leq \frac{11}{9}\text{OPT}(\mathcal{I}) + 4$. Later, a simpler proof by Baker (1985) showed that the additive constant is no more than 3, whilst Yue (1991) improved the bound to $\text{FFD}(\mathcal{I}) \leq \frac{11}{9}\text{OPT}(\mathcal{I}) + 1$. Dósa (2007) then proved that FFD requires at most $\frac{11}{9}\text{OPT}(\mathcal{I}) + \frac{6}{9}$ bins, and that this bound is tight (Dósa et al., 2013). Due to the initial sorting of the items, the time complexity of both FFD and BFD is $O(n \log n)$.

Although seemingly simple and straightforward, heuristics such as FF and FFD form the basis for the development of more complex heuristics. For example, the Refined First-Fit (RFF) algorithm proposed by Yao (1980) has an improved bound of $\frac{5}{3}\text{OPT}(\mathcal{I}) + 5$ on the FF heuristic. Lee and Lee (1985) then presented Harmonic$_M$, which operates by partitioning the items into M subsets, and the Refined-Harmonic algorithm comprising characteristics from both RFF and Harmonic$_M$. A variety of heuristics based on Harmonic$_M$ were later designed, including the Harmonic++ algorithm of Seiden (2002).

Further advanced heuristics for the BPP have been developed with positive results, such as the Minimum Bin Slack (MBS) heuristic (Gupta and Ho, 1999), which focuses on packing each bin in turn rather than each item, and modifications of MBS such as the Perturbation-MBS′ heuristic of Fleszar and Hindi (2002). A comprehensive overview of these heuristics and related methods can be seen in Johnson (1973, 1974), Johnson et al. (1974), Garey et al. (1972), and Coffman et al. (1984, 1996, 1999, 2013).

2.5 Metaheuristics for the Bin Packing Problem

A metaheuristic is a high-level problem-independent framework designed to develop heuristic optimisation algorithms, with the aim of finding the best solution from all possible feasible solutions. This is done by evaluating and iteratively making changes to potential solutions in order to obtain new, superior solutions (Sörensen and Glover, 2013). Metaheuristics are therefore a suitable approach to the BPP as the set $\mathcal{F}$ of feasible bins will be too large to enumerate in all but the most trivial of instances. In this thesis, we focus on one particular class of metaheuristics known as *evolutionary algorithms*, which operate on a population of solutions for a given problem instance.

Natural selection, also known as "survival of the fittest", is defined as the "preservation of favourable individual differences and variations, and the destruction of those that are injurious" (Darwin, 1875). In biological evolution, features that make an individual suitable for the environment tend to be preserved, whilst features that

make the individual weaker are eliminated. Consequently, the fittest individuals are provided more opportunity to breed over the generations, and the superior features of these individuals are transmitted to their offspring during sexual reproduction.

In the 1960s, the notion of natural selection was exploited for use in learning and optimisation problems and led to the development of evolutionary algorithms (EAs), where the biological evolution process is simulated in a computer (Bäck, 1996; Bäck et al., 1997). Within EAs there are three main paradigms: evolution strategies (Schwefel, 1981), genetic algorithms (Holland, 1962a,b, 1975; Goldberg, 1989), and evolutionary programming (Fogel et al., 1966; Fogel, 1995), each created independently with distinct aims. Other types of evolutionary algorithms include genetic programming (Smith, 1980), gene expression programming (Ferreira, 2001), and differential evolution (Storn and Price, 1997).

In general, an evolutionary algorithm is a metaheuristic optimisation algorithm inspired by the natural evolutionary process. Candidate solutions to the problem form the initial population, and procedures emulating selection, reproduction, recombination and mutation are used to create the next generation of solutions (see Chapter 5). This iterative process results in the evolution of the population. Each solution is evaluated based on a specific critera in the form of a fitness function that creates the environment, and solutions which are deemed fitter are provided more opportunity to breed while those which are less so are eliminated.

EAs have been constructed for the one-dimensional bin packing problem in a variety of ways, such as combining EAs with heuristics like FF and MBS′ (Iima and Yakawa, 2003), controlling the characteristics of each solution to be modified or used to form a new solution (Rohlfshagen and Bullinaria, 2007; Quiroz-Castellanos et al., 2015), and randomly partitioning bins in solutions (Falkenauer, 1996). Improved EAs for the BPP have also been addressed: Fan et al. (2020) propose a replacement insertion method designed to enhance the mutation operator, whilst Cardoso Silva and Hasenclever Borges (2019) incorporate heuristics to decode mapped solutions to ensure solution feasibility. In addition, several studies focus on the execution time of the EA, including dividing the population into semi-isolated sub-populations for parallel processing in each iteration (Kucukyilmaz and Kiziloz, 2018), and exploiting the power of the GPU to enable more effective exploraton of the solution space (Ozcan et al., 2016). Beyond the BPP, a wide range of grouping problems have also been tackled using EAs with positive results, including graph colouring (Galinier and Hao, 1999), finding maximum independent sets in graphs (Hifi, 1997), scheduling problems (Kammarti et al., 2013), and other packing problems related to the BPP (Bortfeldt, 2006; Kröger, 1995; Lewis and Holborn, 2017).

In addition to EAs, numerous metaheuristics have been and are continuing to

be developed and applied to various combinatorial optimisation problems. Some of the most famous and influential metaheuristics include tabu search (Glover, 1986), simulated annealing (Kirkpatrick et al., 1983), iterated local search (Lourenço et al., 2003), GRASP (Feo and Resende, 1995), variable neighbourhood search (Mladenović and Hansen, 1997), ant colony optimisation (Dorigo, 1992), and particle swarm optimisation (Kennedy and Eberhart, 1995). An abundance of literature on metaheuristics is available, with notable works by Bianchi et al. (2009), Blum and Roli (2003), Glover and Kochenberger (2003), Mitchell (1996), Sörensen et al. (2017), and Talbi (2009).

2.6 Exact Algorithms for the Bin Packing Problem

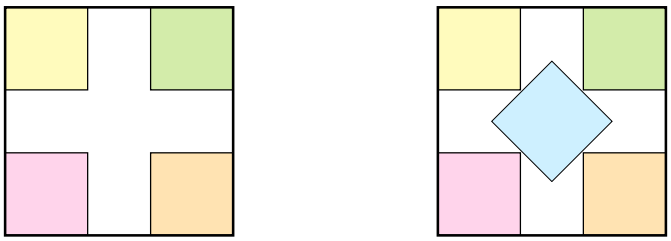
Exact algorithms, whilst being able to produce optimal solutions for problem instances, require a substantial amount of time and computational effort which only increases as the size of the problem instance increases. Nevertheless, there have been many developments into exact algorithms for the BPP.

In 1971, Eilon and Christofides proposed the first exact algorithm using an integer linear programming (ILP) formulation to be solved by the Balas' Additive Algorithm of Balas (1965), a general enumerative scheme for solving linear programs with binary variables. Martello and Toth (1990a) then introduced MTP (Martello-Toth Procedure), a powerful exact algorithm of running time $O(n^2)$ which makes use of the dominance criterion of Martello and Toth (1990b) (discussed in Chapter 5), and also includes heuristics such as FFD. Features of MTP were then exploited for the creation of new exact algorithms. For example, the BISON algorithm of Scholl et al. (1997) combined characteristics from MTP and metaheuristic techniques such as tabu search, whilst Schwerin and Wäscher (1999) used MTP along with new lower bounds obtained from the column generation method of Gilmore and Gomory (1961, 1963). The Bin Completion algorithm, an alternative to MTP, was then presented by Korf (2002, 2003) and later improved by Schreiber and Korf (2013) to be up to five times faster than the original version for problem instances comprising 100 items. A deeper review into these methods and other exact algorithms for the BPP can be found by Delorme et al. (2016).

2.7 Variations of the Bin Packing Problem

Firstly, we note that the majority of the literature on rectangular packing problems are restricted to orthogonal packings, where the edges of the rectangular items must be parallel to the sides of the bin. Although this constraint is necessary for many applications of the BPP, orthogonal packings are not always optimal as can be seen

in Figure 2.4, where an orthogonal packing only permits four of the items to be packed, whilst a non-orthogonal packing accommodates five items.



(a) (b)

Figure 2.4: (a) An orthogonal packing, where only four items of equal size can be packed; and (b) a non-orthongonal packing where a fifth item can be packed into the centre of the bin.

The BPP forms the basis of many packing problems involving additional constraints on the items and bins: Haouari and Serairi (2009) addressed the variable-sized BPP using bins of unequal capacities, whilst Zhou et al. (2009) extended the problem where, in addition to variable-sized bins, items and bins are assigned types, and items can only be packed into bins of the same type. Lodi et al. (1999) presented approximation algorithms for a two-dimensional BPP in which the items cannot be rotated and must be packed in their given orientation. There also exists a dual version of the BPP which, given a positive constant C, involves packing items into a *maximum* number of bins of capacity W such that the total width of items in each bin is at least C (Csirik and Totik, 1988). Furthermore, problems can consist of non-rectangular items and non-rectangular bins. For example, in Akeb et al. (2009), a set of non-identical circular items are to be packed into a larger circle such that the radius of the circle is minimised.

One particular problem involving non-rectangular items is the Trapezoid Packing Problem (TPP) initially investigated by Lewis et al. (2011), where trapezoid-shaped items are to be packed into bins so as to minimise the number of bins required whilst also attempting to reduce the amount of triangular waste between adjacent trapezoids. Although the TPP itself is NP-hard, the problem of determining a feasible arrangement of trapezoids into a single bin has been shown to be solvable in polynomial-time (Lewis and Holborn, 2017). Figure 2.5 shows two possible feasible arrangements of five trapezoids in a single bin.

There are also special cases of the BPP with characteristics that allow for a solution to be obtained in polynomial-time. The most trivial type of problem involves packing items of equal size into bins of equal capacity, where a heuristic such as FF will yield an optimal solution. Coffman et al. (1987) studied sets of items with divisible items sizes, showing that not only is FFD optimal for such instances, but even FF can produce optimal solutions if the largest item size in the set divides the

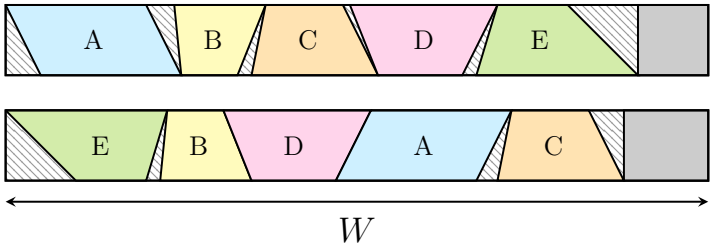


Figure 2.5: Two feasible alignments for a set of five trapezoids, where the lined areas are the triangular inter-item waste and the remaining unused space at the ends of the bins are shaded in grey.

bin capacity. Moreover, Fernandez de la Vega and Lueker (1981) proved that there exists a polynomial-time algorithm for solving instances of the BPP in which there are at most l different sizes of items and at most m items can be packed into a single bin.

There are a number of well-known problems related to the BPP that have particular applications in various industries, such as the knapsack problem, the cutting stock problem, and the strip packing problem. A comprehensive classification of one- and multi-dimensional cutting and packing problems can be found by Wäscher et al. (2007).

2.7.1 The Knapsack Problem

The knapsack problem can be viewed as a single-bin version of the BPP, where the objective is to be maximised rather than minimised. Given a set $\mathcal{I}$ of n items of varying weights $w_i \in \mathbb{Z}^+$ and values $v_i \in \mathbb{Z}^+$ for all $i \in \mathcal{I}$, and a knapsack of weight capacity W, the problem involves finding a subset of items $\mathcal{I}' \subseteq \mathcal{I}$ such that the total weight of the items in $\mathcal{I}'$ does not exceed the knapsack capacity W and the total value of the items in $\mathcal{I}'$ is maximised (Martello and Toth, 1990a).

$$\text{maximise} \quad \sum_{i=1}^{n} v_i x_i \tag{2.4a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq W \tag{2.4b}$$

$$x_i \in \{0,1\} \qquad i = 1, \ldots, n. \tag{2.4c}$$

The objective (2.4a) is to maximise the value of the items in the knapsack. Constraint (2.4b) ensures the total weight of the items in the knapsack does not exceed the capacity W, and Constraint (2.4c) states that the variables must be binary, i.e. each item in $\mathcal{I}$ is either chosen to be in the knapsack once or is not chosen at all.

The knapsack problem, which can be traced back to Mathews (1896), has applications in resource allocation and security as well as investment. For example, when faced with a set of projects, each with varying amounts of initial costs and potential profits, an investor with a fixed amount of capital must decide which projects to invest in to procure the largest profit. As with the BPP, the knapsack problem is NP-hard.

2.7.2 The Cutting Stock Problem

The cutting stock problem (CSP) is as follows: an order consists of m items types, each with size $w_i \in \mathbb{Z}^+$ and demand $d_i \in \mathbb{Z}^+$ for $i = 1, \ldots, m$. An unlimited number of stocks are available of equal size W and some cost. The task involves cutting d_i copies of each item type i from the fewest number of stocks, thus fulfilling the order at the least cost.

$$\text{minimise} \quad \sum_{j=1}^{n} y_j \tag{2.5a}$$

$$\text{subject to} \quad \sum_{i=1}^{m} w_i x_{ij} \leq W y_j \qquad j = 1, \ldots, n \tag{2.5b}$$

$$\sum_{j=1}^{n} x_{ij} = d_i \qquad i = 1, \ldots, m \tag{2.5c}$$

$$y_j \in \{0, 1\} \qquad j = 1, \ldots, n \tag{2.5d}$$

$$x_{ij} \geq 0, \text{ integer} \qquad i = 1, \ldots, m, j = 1, \ldots, n. \tag{2.5e}$$

Here, the variable x_{ij} denotes the number of items of type i that are cut from stock j. Similarly to the BPP, the objective (2.5a) is to use the fewest number of stocks $j = 1, \ldots, n$ (where n is some large number), and Constraint (2.5b) ensures that for each stock the total size of the items does not exceed the size of the stock. Constraint (2.5c) imposes the condition that all demands for each item are satisfied, and Constraints (2.5d) and (2.5e) state the variable requirements.

The BPP is a special case of the CSP where the demand of each item type $d_i = 1$ for all item types $i = 1, \ldots, m$. It can be seen then that the CSP mainly differs from the BPP in that the items tend to be weakly heterogeneous – that is, many items have the same dimensions. Consequently, the problem typically involves producing "patterns", possible arrangements of items on a single piece of stock. A solution then comprises a set of patterns repeated over multiple pieces of stock. The most notable work on the CSP is that of Gilmore and Gomory (1961, 1963), showing that a column generation approach can be used in place of calculating all possible patterns, and is guaranteed to converge to an optimal solution. There exists an abundance

of literature investigating the CSP with alternative constraints, such as generating solutions that produce useable leftover material (Coelho et al., 2017; do Nascimento et al., 2020), and incorporating leftover material in new solutions (Cui et al., 2017), which have a desirable environmental impact, and finding solutions using a minimal number of patterns so as to reduce setup costs (Wu and Yan, 2016; Ma et al., 2019), as well as dealing with multiple stock lengths (Belov and Scheithauer, 2002; Poldi and Arenales, 2009). In particular, the two-dimensional CSP involving irregular-shaped items has been well documented, using approaches such as column generation and sequential heuristic procedures (Song and Bennell, 2014), branch and bound (Alvarez-Valdes et al., 2013), and the "no-fit polygon" concept (Bennell et al., 2001).

An interesting extension of the CSP described by Garraffa et al. (2016) considers sequence-dependent cut losses (SDCL). Here, the type of machine used results in additional material loss, known as a "cut loss", c_{ij} between items i and j during the cutting process. The amount of loss can vary between different items and is also dependent on the order of the items: a cut loss between two adjacent items A and B with A packed first may not necessarily be equal to the cut loss that arises when B is packed first. There are also cut losses c_{0i} and c_{i0} that occur before the first item and after the last item than must be taken into account. Hence, the CSP-SDCL involves arranging the items onto the fewest pieces of stock such that the sum of the item widths *and* the sum of the cut losses between all adjacent items does not exceed the stock width W. Figure 2.6 shows an instance of the CSP-SDCL comprising four item types with their respective widths w_i and demand d_i, the corresponding cut losses matrix, and a feasible solution consisting of three pieces of stock material. Observe that this problem can also be viewed as a vehicle routing problem, where the vehicles represent the stock material, the cities represent the items, and the cut losses matrix shows the distances between each pair of cities.

2.7.3 The Strip Packing Problem

The strip packing problem (SPP) is a two-dimensional packing problem classified as an open-dimensional problem by Wäscher et al. (2007). Given a set $\mathcal{I}$ of n rectangular items of varying heights $h_i \in \mathbb{Z}^+$ and widths $w_i \in \mathbb{Z}^+$ for all $i \in \mathcal{I}$, and a strip of fixed width W and infinite height H, the problem involves packing the items onto the strip such that no items overlap and the height H of the items on the strip is minimised.

Consider the bottom-left corner of the strip as the origin of the xy-plane, where the width of the strip is along the x-axis and the height of the strip is along the y-axis. Then, the coordinates (x_i, y_i) represent the location of the bottom-left corner

i	A	B	C	D
w_i	2.5	3	2	4
d_i	2	2	2	1

$$\begin{bmatrix} 0 & 2.1 & 0.4 & 0.3 & 1.2 \\ 0.7 & 3.2 & 0.6 & 1.8 & 2.7 \\ 0.4 & 1.5 & 3.1 & 2 & 1.3 \\ 1.1 & 1 & 2.9 & 3.4 & 2.2 \\ 2.8 & 0.5 & 1.4 & 1.9 & 0.6 \end{bmatrix}$$

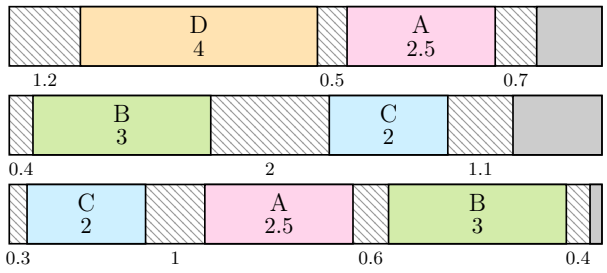


Figure 2.6: An example instance and solution for the CSP-SDCL showing the widths w_i and demand d_i for each item type i and the cut losses matrix. The solution comprises three pieces of stock, where the lined areas are the cut losses and the remaining space on the end of each stock is shaded in grey. Here, $W = 10$.

of each item $i \in \mathcal{I}$ on the strip. From this, we can formulate the SPP as follows:

$$\begin{aligned}
\text{minimise} \quad & H && && \text{(2.6a)} \\
\text{subject to} \quad & x_i + w_i \leq W && i = 1, \ldots, n && \text{(2.6b)} \\
& y_i + h_i \leq H && i = 1, \ldots, n && \text{(2.6c)} \\
& x_i + w_i \leq x_j \text{ or } x_j + w_j \leq x_i \text{ or} && && \\
& y_i + h_i \leq y_j \text{ or } y_j + h_j \leq y_i && i, j = 1, \ldots, n,\ i \neq j && \text{(2.6d)} \\
& x_i, y_i \geq 0 && i = 1, \ldots, n. && \text{(2.6e)}
\end{aligned}$$

Clearly, the objective (2.6a) is to reduce the overall height on the packing on the strip. Constraints (2.6b), (2.6c), and (2.6e) impose the requirement for all items to be packed within the bounds $H \times W$ of the strip. Constraint (2.6d) ensures that no items overlap (Kenmochi et al., 2009).

The SPP is a generalisation of the BPP, where in the BPP all heights are equal; thus concluding that the SPP is also NP-hard. From the original problem, introduced by Baker et al. (1980), stems an abundance of adaptations developed for a variety of applications. One additional constraint to the SPP involves packing the items in their given orientations (Ntene and van Vuuren, 2008), whilst much literature exists for the SPP in which the rectangular items can be rotated 90° – see, for example, the works of Jansen and van Stee (2005), Kenmochi et al. (2009), Cui et al. (2013), and He et al. (2013). An example of two solutions for an instance of the SPP with and without item rotations is provided in Figure 2.7, showing that the ability to rotate the items can reduce the height of the packing.

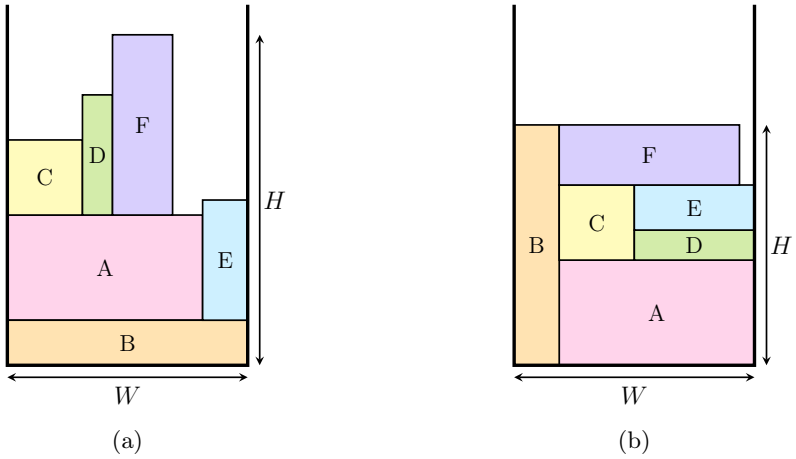


Figure 2.7: Feasible solutions for an example instance of the SPP comprising 6 items. In (a) the items are packed in their given orientations, whilst in (b) the items can be rotated by 90°, resulting in a better packing.

Further constraints have been added to SPPs such as the "guillotineable" requirement, where items are to be packed onto the strip such that, at specific locations, a straight-line cut can be made across the width of the strip without cutting through an item as shown in Figure 2.8 (Kröger, 1995; Hifi, 1998). Other related problems include shelf divisions, in which items must be packed into subsections on the strip (Xavier and Miyazawa, 2008), and unloading constraints where the feasibility of the solution is dependent on the order of the items on the strip (da Silveira et al., 2013).

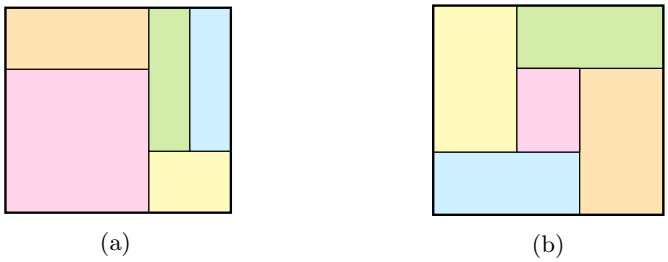


Figure 2.8: Examples of (a) a guillotineable packing; and (b) a non-guillotineable packing.

2.8 The Score-Constrained Packing Problem

We now turn our attention to the Score-Constrained Packing Problem (SCPP). Let $\mathcal{I}$ be an instance of the SCPP, which contains $|\mathcal{I}| = n$ rectangular items of fixed height $H > 0$ and varying widths w_i and score widths a_i, b_i for each item $i \in \mathcal{I}$, as described in Definition 1.1. Then, given a minimum scoring distance $\tau \in \mathbb{Z}^+$, a feasible solution for an instance $\mathcal{I}$ of the SCPP is represented by the set $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ such

that:

$$\bigcup_{j=1}^{k} S_j = \mathcal{I}, \tag{2.7a}$$

$$S_i \cap S_j = \emptyset \qquad i, j = 1, 2, \ldots, k,\ i \neq j, \tag{2.7b}$$

$$A(S_j) = \sum_{i=1}^{|S_j|} w_i \leq W \quad j = 1, 2, \ldots, k, \tag{2.7c}$$

$$\mathbf{r}(i) + \mathbf{l}(i+1) \geq \tau \qquad i = 1, 2, \ldots, |S_j| - 1,\ j = 1, 2, \ldots, k. \tag{2.7d}$$

In other words, the solution $\mathcal{S}$ must contain all items in $\mathcal{I}$ (2.7a), each item in $\mathcal{I}$ must be packed into exactly one of the bins in $\mathcal{S}$ (2.7b), the total width of the items in each bin cannot exceed the bin's capacity W (2.7c), and the items must be arranged such that the vicinal sum constraint (1.1) is fulfilled in each bin (2.7d). Here, a bin $S_j \in \mathcal{F}$ if and only if Constraints (2.7c) and (2.7d) are satisfied. An optimal solution for the SCPP is a solution comprising the fewest number of bins required to feasibly pack all items in $\mathcal{I}$; thus the aim is to minimise the number of bins k. Observe that the BPP can be seen as a special case of the SCPP where $\tau = 0$, as the vicinal sum constraint will always be satisifed. As a result, the SCPP is also NP-hard.

As previously mentioned, there exists a basic lower bound t (2.2) for the number of bins k in a solution for an instance of the BPP. However, this lower bound does not perform as accurately for the SCPP on account of the vicinal sum constraint. Consider an instance $\mathcal{I}$ of n items in which the largest score width of all $2n$ score widths is less than $\tau/2$. Evidently, none of the items can be placed next to one another as there are no pairs of score widths that comply with the vicinal sum constraint. Under these circumstances, each item must be packed into individual bins; thus $k = n$. The theoretical minimum t for the BPP does not take into consideration the effect of the minimum scoring distance τ on the feasibility of the solution. Therefore, a solution for an instance of the SCPP comprising more than t bins could in fact be optimal, however as there is currently no appropriate lower bound for the SCPP it would be difficult to determine the optimality of a solution for large instances in a reasonable amount of time. Nevertheless, if a solution for an instance of the SCPP comprises exactly t bins then the solution is guaranteed to be optimal.

In addition to the lower bound, the vicinal sum constraint also gives rise to further differences in solutions for the BPP and SCPP. Obviously, the main distinction involves the ordering and orienation of the items in the bins: inconsequential in the BPP, yet crucial for the feasibility of a solution for the SCPP. This subsequently introduces a disparity when amending solutions. A solution to the BPP remains feasible when an item is removed from a bin or when an item is added to a bin,

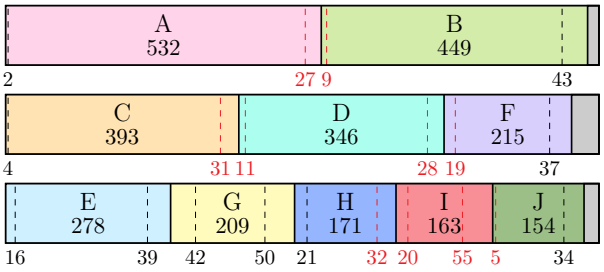
provided the bin can accommodate the item. In contrast, this may render a solution for the SCPP infeasible as the new adjacent score widths may not satisfy the vicinal sum constraint.

In essence, the SCPP involves deciding (a) which items should be packed into which bins and (b) the order and orientation of each item within each bin. This problem characteristic also occurs in two of the problems discussed in the previous section, namely the TPP and the CSP-SDCL. One particular difference, however, between these problems and the SCPP concerns the feasibility of individual bins. For example, in the TPP it is still possible to pack trapezoids with opposite angles, i.e. '\' and '/', alongside one another, despite the unnecessarily large inter-item waste. Similarly, in the CSP-SDCL two items with a large cut loss between them can still be packed alongside one another if required. Both of these packing problems permit *any* order and orientation of items provided the bins are not overfilled. Conversely, the SCPP possesses the strong vicinal sum constraint, which if violated immediately causes an alignment of items in a bin to be invalid, rendering the entire solution infeasible.

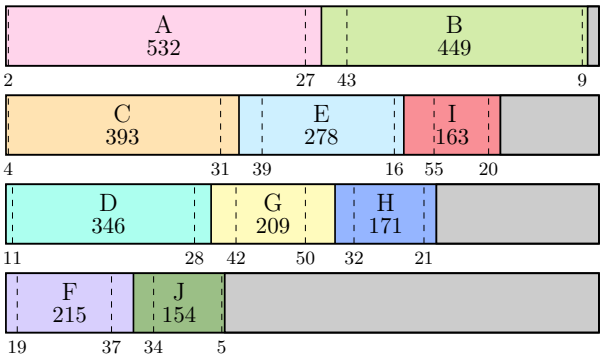
To illustrate the differences between the BPP and the SCPP, suppose we use items for an instance $\mathcal{I}$ of the SCPP in the previous BPP solution in Figure 2.3, as shown in Figure 2.9.[1] Note that the BPP solution is actually produced using the FFD heuristic described in Section 2.4. As the vicinal sum constraint is not a factor in the BPP and the FFD heuristic does not have the ability to rotate items, all the items are packed into the bins in regular orientations (i.e. with the smallest score width on the left-hand side of the item). Despite using fewer bins, the solution produced for the BPP is not feasible for the SCPP; the vicinal sum constraint is violated at least once in every bin. Furthermore, the theoretical minimum for the instance $\mathcal{I}$ is $t = 3$ which is attainable for the conditions of the BPP (for example, if $\tau = 0$) however, for the SCPP it is known that for this particular instance an optimal solution comprises four bins.

Evidently, basic methods designed for the BPP cannot be used for the SCPP as there is no guarantee that the final solution will be feasible. The disparities between the BPP and SCPP leads us to adapt existing methods and seek new unique approaches capable of producing feasible solutions that adhere to the constraints of the SCPP.

[1]Note that the SCPP instance $\mathcal{I}$ used in Figure 2.9 is the instance depicted in Figure 1.3a in Chapter 1, which is equivalent to the BPP instance in Figure 2.3a with added score widths.



(a) BPP



(b) SCPP

Figure 2.9: A comparison of solutions for the BPP and SCPP using the same problem instance for the SCPP, where the red score lines on the BPP solution show the vicinal sum constraint violations. Here, $|\mathcal{I}| = 10$, $\tau = 70$, and $W = 1000$.

2.9 Summary

In this chapter, we introduced grouping problems and focused on a particular class of such problems known as cutting and packing problems. The one-dimensional bin packing problem (BPP) was presented and shown to be NP-hard, implying that finding optimal solutions for larger problem instances can take an infeasible amount of time. From this, we discussed various approaches to the BPP such as heuristics and metaheuristics which find near-optimal solutions in a more practical time period, as well as a few exact methods that are suitable for smaller, more manageable problem instances. The BPP was also shown to give rise to numerous alternative cutting and packing problems that have been adapted to suit a range of applications.

By reviewing the BPP, we were able to observe the similarities and differences between the BPP and the Score-Constrained Packing Problem (SCPP), and deduce that the SCPP is also NP-hard. The most notable contrast is the ordering and orientation implications associated with the SCPP, indicating that methods available for the BPP are incompatible with the SCPP. Consequently, the remainder of this thesis is devoted to constructing algorithms for the SCPP that yield high quality feasible solutions.

Recall that for the TPP – another packing problem involving the ordering and ori-

entation of items (see Section 2.7) – the authors developed an exact polynomial-time algorithm for determining a feasible alignment of items in a single bin. Following on in a similar manner, the next chapter presents an algorithm for the Score-Constrained Packing Sub-Problem (sub-SCPP) introduced in Definition 1.2.

Chapter 3

The Score-Constrained Packing Sub-Problem

3.1 Introduction

As described in Definition 1.2, the Score-Constrained Packing Sub-Problem (sub-SCPP) involves packing a subset $\mathcal{I}' \subseteq \mathcal{I}$ of items into a single bin such that the vicinal sum constraint (1.1) is fulfilled. Given n cities and distances between each pair of cities, the travelling salesman problem (TSP) involves determining the shortest route that visits each city exactly once and returns to the origin city. In previous literature, the sub-SCPP has been modelled as a generalised TSP in various ways.

Goulimis (2004) suggested considering the sub-SCPP as a generalised TSP known as the travelling politician problem (TPP), in which a politician must visit exactly one of two cities in every state. However, rather than attempting to find the shortest tour, the aim is to determine whether such a tour exists.

This approach can be formalised as follows (Becker, 2010): given a set of n items, each item is assigned two vertices on a graph G. These vertices denote the two different orientations of the item, where in the set $V = \{v_1, v_2, \ldots, v_{2n}\}$ the odd-indexed vertices represent each item in a regular orientation, whilst the even-indexed vertices represent each item in a rotated orientation. Let $w_1(v_i)$ and $w_2(v_i)$ be the left and right score widths of regular items for all odd-indexed vertices $i = 1, 3, \ldots, 2n-1$, and let $w_1(v_i)$ and $w_2(v_i)$ be the right and left score widths of rotated items for all even-indexed vertices $i = 2, 4, \ldots, 2n$. Then, $w_1(v_{2k-1}) = w_2(v_{2k})$ and $w_2(v_{2k-1}) = w_1(v_{2k})$ for all $k = 1, 2, \ldots, n$.

Given a minimum scoring distance $\tau \in \mathbb{Z}^+$, the vicinal sum constraint is satisfied for two vertices $v_i, v_j \in V$ that do not represent the same item if and only if $w_1(v_i) + w_2(v_j) \geq \tau$ or $w_2(v_i) + w_1(v_j) \geq \tau$. Two constraints are required to account for the different orderings of the items, i.e. whether the item represented by v_i comes before

or after the item represented by v_j when packing from left to right. As each vertex on G has two weights, we need to ensure that each weight is only used once in the final cycle. That is, if a vertex v_i is adjacent to the previous vertex in the cycle v_{i-1} by its weight $w_1(v_i)$, then v_i must be adjacent to the next vertex v_{i+1} by its weight $w_2(v_i)$, or vice versa. Therefore, a record must be kept of which weights are being used for adjacency between every pair of vertices. To do this, we add a directed edge set E to G:

$$E = \{(v_i, v_j) : w_2(v_i) + w_1(v_j) \geq \tau,\ i = 2k-1\ \wedge j \neq 2k \\ \vee\ i = 2k \wedge j \neq 2k-1\ \forall\ k = 1, 2, \ldots, n\}.$$

A dummy vertex is added to the graph which represents the "base city" from which the tour will begin and end. An example of this model is shown in Figure 3.1, where the pair of vertices for each item can be seen as two cities in each state.

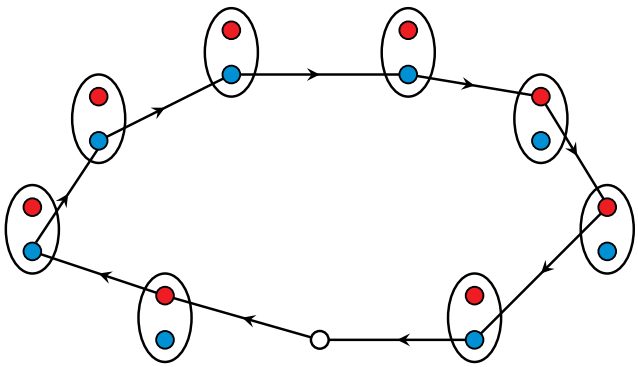
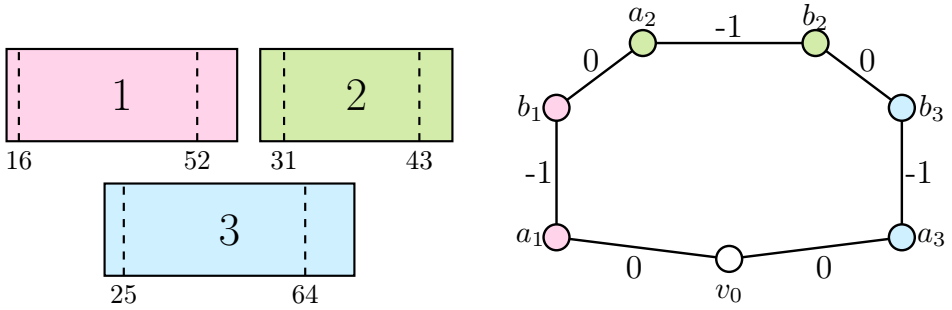


Figure 3.1: The sub-SCPP modelled as a travelling politician problem, with red and blue vertices representing each item in their regular and rotated orientations respectively.

For the TPP, the task is to find a tour in which the politician visits exactly one city in each state before returning to the base city. In other words, given the graph G, the problem involves deciding whether there exists a cycle in G that contains exactly one vertex out of each of the vertex subsets $\{2k-1, 2k\}\ \forall\ k = 1, 2, \ldots, n$. Using this model requires two vicinal sum constraints to be assessed as each orientation of every item is represented by an individual vertex, however the final solution to the problem will comprise just half of the vertices on the graph. Note also that for every directed cycle in G there is corresponding directed cycle in the opposite direction due to symmetry, i.e. a tour in which the cities are visited in reverse order using the other vertex in each subset. As the TPP is a generalisation of the TSP, it follows that this problem is NP-hard.

An alternative to the TPP model was presented by Lewis et al. (2011). Again, each of the n items are assigned two vertices, however these vertices correspond to the score widths of the item rather than the item's orientations. The graph G, which also includes a dummy vertex, is a complete graph; thus every pair of vertices

is connected by an edge. Edges between vertices that represent score widths on the same item have weight -1, and edges between vertices that represent score widths on different items with total width less than or equal to τ have weight ∞. All other edges have weight 0. Thus, the problem can be seen as a TSP, where starting from the origin city (the dummy vertex) a tour must be found that has distance exactly $-n$. Figure 3.2 depicts an example of this model.



(a) $n = 3$ items

(b) A valid tour

Figure 3.2: An example model of the sub-SCPP as generalised TSP suggested by Lewis et al. (2011), where a tour of length $-n$ has been found.

This model has an advantage over Goulimis' TPP model in that all the vertices on the graph will be used, and fewer edge calculations are required as there are no duplicate edges between vertices. A more intuitive model introduced by Becker (2010) further simplifies the above model by maintaining two distinct edge sets. It is this version that we will use to model instances of the sub-SCPP.

3.2 Modelling the sub-SCPP

Firstly, consider the following sequencing problem originally presented by Hawa et al. (2018):

Definition 3.1 *Let $\mathcal{M}$ be a multiset of unordered pairs of integers $\mathcal{M} = \{\{a_1, b_1\}, \{a_2, b_2\}, \ldots, \{a_n, b_n\}\}$, and let $\mathcal{T}$ be a sequence of the elements of $\mathcal{M}$ in which each pair is a tuple. Given a fixed value $\tau \in \mathbb{Z}^+$, the* Constrained Ordering Problem (COP) *consists in finding a solution $\mathcal{T}$ such that the sum of adjacent values from different tuples is greater than or equal to τ.*

For example, given the instance $\mathcal{M} = \{\{4, 21\}, \{9, 53\}, \{13, 26\}, \{17, 29\}, \{32, 39\}, \{35, 41\}, \{44, 57\}, \{48, 61\}\}$ and $\tau = 70$, one possible feasible solution is $\mathcal{T} = \langle(4, 21), (53, 9), (61, 48), (26, 13), (57, 44), (32, 39), (35, 41), (29, 17)\rangle$.

By viewing each pair in $\mathcal{M}$ as an item i represented by its score widths a_i, b_i, we see that the COP is in fact equivalent to the sub-SCPP. Then, the requirement for the sum of adjacent values to exceed τ corresponds to the vicinal sum constraint

(1.1), where τ is the minimum scoring distance. Figure 3.3 shows an instance of the sub-SCPP which is equal to the example problem instance $\mathcal{M}$ of the COP stated above. In the sub-SCPP, the provided set of items $\mathcal{I}' \subseteq \mathcal{I}$ has total width that is able to fit into a single bin, i.e. $A(\mathcal{I}') \leq W$, and so the items' individual widths do not need to be taken into consideration when seeking a feasible alignment. Thus, any instance of the sub-SCPP can be simplified by being transformed into an instance of the COP. In the following, we shall be focusing solely on the COP.

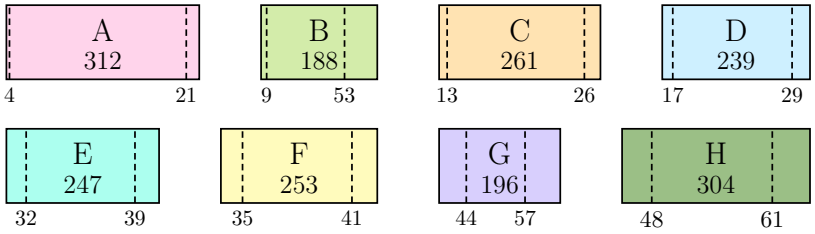
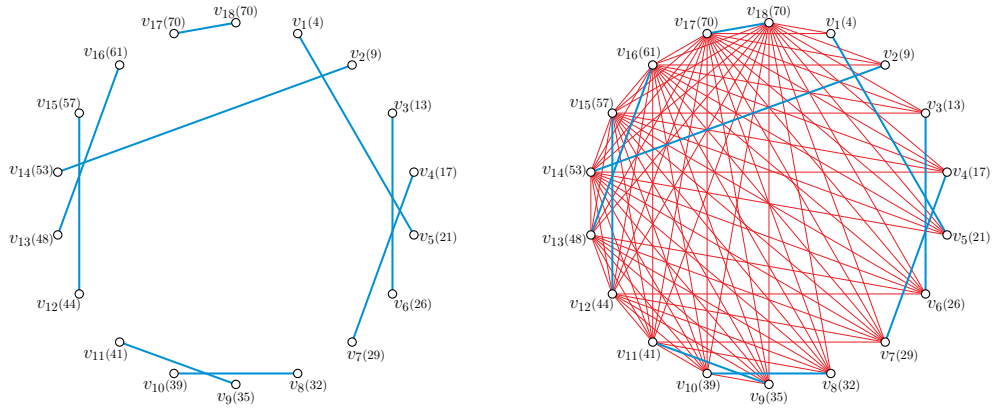


Figure 3.3: An instance $\mathcal{I}$ of the sub-SCPP comprising eight items. This instance is equivalent to the COP instance $\mathcal{M} = \{\{4, 21\}, \{9, 53\}, \{13, 26\}, \{17, 29\}, \{32, 39\}, \{35, 41\}, \{44, 57\}, \{48, 61\}\}$.

As with Goulimis' and Lewis' approaches, it is useful to model an instance $\mathcal{M}$ of the COP graphically (Hawa et al., 2018; Becker, 2010). For every pair $\{a_i, b_i\} \in \mathcal{M}$, two vertices u, v with weights $w(u) = a_i$, $w(v) = b_i$ are formed, together with a "blue" edge $\{u, v\}$. Such vertices created from a pair in $\mathcal{M}$ and connected by a blue edge are referred to as *partners*. This gives a vertex-weighted graph G with $2n$ vertices and n edges. Without loss of generality, we assume that the vertices $\{v_1, \ldots, v_{2n}\}$ are labelled in weight order such that $w(v_i) \leq w(v_{i+1})$. An extra pair of partner vertices, v_{2n+1}, v_{2n+2}, with weights $w(v_{2n+1}) = w(v_{2n+2}) = \tau$ is also added to G, along with a blue edge $\{v_{2n+1}, v_{2n+2}\}$. The edge set B contains all blue edges between partners. By denoting the partner of a vertex v_i as $p(v_i)$, the set B can be defined as $\{\{v_i, p(v_i)\} : v_i \in V\}$. Figure 3.4a shows the graph $G = (V, B)$ comprising $n + 1$ components based on the above example instance $\mathcal{M}$ of the COP.

A second set of edges, R, is added to G, which contains "red" edges between vertices that are not partners and whose combined weight equals or exceeds τ; that is, $R = \{\{v_i, v_j\} : w(v_i) + w(v_j) \geq \tau \ \wedge \ p(v_i) \neq v_j \ \forall \ v_i, v_j \in V, v_i \neq v_j\}$. It follows that B and R are disjoint edge sets, i.e. $B \cap R = \emptyset$. Figure 3.4b shows the resulting graph $G = (V, B \cup R)$ modelling our example instance $\mathcal{M}$. The additional vertices, v_{2n+1} and v_{2n+2}, are in fact *universal* vertices with $\deg(v_{2n+1}) = \deg(v_{2n+2}) = 2n+1$, as they are adjacent to every other vertex via an edge in R due to their large weights. Observe the pattern on the graph G, with $\deg(v_i) \leq \deg(v_{i+1})$ as the vertices are in weight-ascending order.

Consider the following definition for graphs with a specific structure (Chvátal and Hammer, 1977):



(a) $G = (V, B)$ (b) $G = (V, B \cup R)$

Figure 3.4: The graph $G = (V, B \cup R)$ modelling our example instance $\mathcal{M}$ of the COP. Here, thicker blue edges are in B and thinner red edges are in R, with the vertices' weights stated in parentheses.

Definition 3.2 *Let $G = (V, E)$ be a simple, undirected graph with a weight function $w : V \to \mathbb{Z}^+$ such that each vertex has weight $w(v_i)$ $\forall$ $v_i \in V$, and let $\tau \in \mathbb{Z}^+$ be a "threshold" value. Then, G is a* threshold graph *if*

$$E = \{\{v_i, v_j\} : w(v_i) + w(v_j) \geq \tau \ \forall \ v_i, v_j \in V, v_i \neq v_j\}, \tag{3.1}$$

that is, an edge exists between two distinct vertices if the sum of their weights is greater than or equal to the threshold value.

Threshold graphs, initially defined by Chvátal and Hammer (1973), have been widely studied due to their unique structure, most notably the degree sequence of the graphs (Koren, 1973) as well as the split graph property where the vertex set can be partitioned into a stable set and a clique. This has led to many useful applications of threshold graphs for problems such as parallel processing and scheduling (Ecker and Zaks, 1977; Henderson and Zalcstein, 1977). A comprehensive overview of threshold graphs is provided by Mahadev and Peled (1995), and further research is available in addition to the above literature by Golumbic (2004), Orlin (1977), and Brandstadt et al. (1999).

From this, it can be seen that our graph $G = (V, B \cup R)$ is a type of modified threshold graph, where R contains edges between vertices whose total weight meets the threshold value τ provided the vertices are not partners. This underlying threshold graph characteristic will be useful in our approach to solving the COP.

3.2.1 Hamiltonian Cycles

Definition 3.3 *A* Hamiltonian cycle *in a graph G is a cycle that visits every vertex of G exactly once. A graph containing a Hamiltonian cycle is said to be Hamiltonian.*

The Hamiltonian cycle problem (HCP), determining whether a given graph is Hamiltonian, is NP-complete for both undirected and directed graphs (Karp, 1972), whilst the problem of actually finding a Hamiltonian cycle is NP-hard. There are $n!$ sequences of vertices in an n-vertex graph that could form a Hamiltonian cycle; given large graphs it would not be possible to assess all sequences in a realistic time period. The difficulty of the HCP has led to much research into characteristics of Hamiltonian graphs and criertia for the existence of Hamiltonian cycles (Gould, 1991, 2003) as well as methods of solving the HCP. Notable methods include an exact enumerative algorithm (Martello, 1983), a search procedure using partial paths (Rubin, 1974), a Monte Carlo algorithm involving cycle covers and the inclusion-exclusion principle (Bjorklund, 2014), and dynamic programming approaches (Bellman, 1961; Held and Karp, 1962).

The HCP on a graph $G = (V, E)$ is a special case of the TSP, where the TSP has a set C of cities identical to V (where $|V| = n$). Then, for every pair of cities $v_i, v_j \in C, v_i \neq v_j$, the distance $d(v_i, v_j)$ between the two cities is 1 if $\{v_i, v_j\} \in E$ and 2 if $\{v_i, v_j\} \notin E$. Then, G contains a Hamiltonian cycle if and only if there exists a tour in the equivalent TSP that has length n, otherwise the tour will be longer (Garey and Johnson, 1979).

As the values in $\mathcal{M}$ must form a single sequence in the final solution, it follows that all vertices on our graph G modelling the instance of the COP must be in a single cycle that represents a feasible solution; thus, we require a Hamiltonian cycle. Although the order of values within each pair in $\mathcal{M}$ can be rearranged the values themselves cannot be modified, and so every pair of values in $\mathcal{M}$ must remain as a pair in the solution $\mathcal{T}$. Therefore, we need to find a Hamiltonian cycle in which each vertex is preceeded by or succeeded by its partner in the cycle. From this, we introduce a specific type of Hamiltonian cycle:

Definition 3.4 *Let $G = (V, B \cup R)$ be a simple, undirected graph where each edge is a member of one of two sets, B or R. G contains an* alternating Hamiltonian cycle *if there exists a Hamiltonian cycle whose successive edges alternate between sets B and R.*

Such graphs containing two distinct edges sets can be considered as edge-coloured graphs, and so successive edges in an alternating cycle would be of different colours. General alternating cycles have been widely explored in a variety of graphs: Daykin (1976) studied the length of such cycles in complete graphs, whilst Bang-Jensen and Gutin (1998) focused on determining the longest cycle in complete multigraphs. Specifically considering alternating *Hamiltonian* cycles in graphs, Bankfalvi and Bankfalvi (1968) provided a criterion for the existence of an alternating Hamilto-

nian cycle in a 2-edge coloured graph with an even vertex set. In addition, Bollobás and Erdös (1976) describe how the colours of edges incident to each vertex determines the alternating Hamiltonicity of a complete graph, which Chen and Daykin (1976) extended to complete bipartite graphs. Further research includes, for example, that of Grossman and Häggkvist (1983) showing that an alternating Hamiltonian cycle exists if the edges of the graph can be partitioned into two sets such that every vertex on the graph is incident to an edge in both sets, perfect matchings incorporated into alternating Hamiltonian cycles (Häggkvist, 1977; Yang, 1999), as well as a deeper investigation into graphs comprising more than two edge colours (Hilton, 1992). A survey on the abundance of literature on alternating cycles in graphs is provided by Bang-Jensen and Gutin (1997).

3.2.2 Approaching the Constrained Ordering Problem

Recall that, on our graph G, the set B of blue edges between partners represents each pair of values in $\mathcal{M}$, and the red edges in R show all values from different pairs that meet the vicinal sum constraint. Thus, an alternating Hamiltonian cycle in G corresponds to a feasible solution $\mathcal{T}$ for an instance $\mathcal{M}$ of the COP, as explained in the following theorem:

Theorem 3.5 *Let $G = (V, B \cup R)$ be a simple, undirected graph modelling an instance $\mathcal{M}$ of the COP. Then, a feasible solution $\mathcal{T}$ for the given instance $\mathcal{M}$ exists if and only if G contains an alternating Hamiltonian cycle.*

Proof. Let G contain an alternating Hamiltonian cycle. Then, the alternating Hamiltonian cycle corresponds to a legal sequence of the elements in $\mathcal{M}$, as the edges in B in the cycle represent each pair of values in $\mathcal{M}$, and the edges from R in the cycle depict the values from different pairs that meet the vicinal sum constraint and can be ordered so that they are adjacent to one another. Alternatively, let $\mathcal{T}$ be a feasible solution for an instance $\mathcal{M}$ of the COP. It is clear that the vertices corresponding to the pairs of values in $\mathcal{T}$ will be connected by blue edges on G, and the vertices representing neighbouring values from different tuples in $\mathcal{T}$ will be connected by red edges as they meet the vicinal sum constraint. The two universal vertices on G will be adjacent to the vertices representing the outermost values of the sequence $\mathcal{T}$. Therefore, G will contain an alternating Hamiltonian cycle. □

Figure 3.5 shows an alternating Hamiltonian cycle in the graph G corresponding to a feasible solution $\mathcal{T}$.

A *matching* in a graph G is a set of pairwise non-adjacent edges, i.e. no two edges share a common vertex. A matching is said to be *maximum* if it contains the largest possible number of edges. A *perfect* matching is a matching in which every

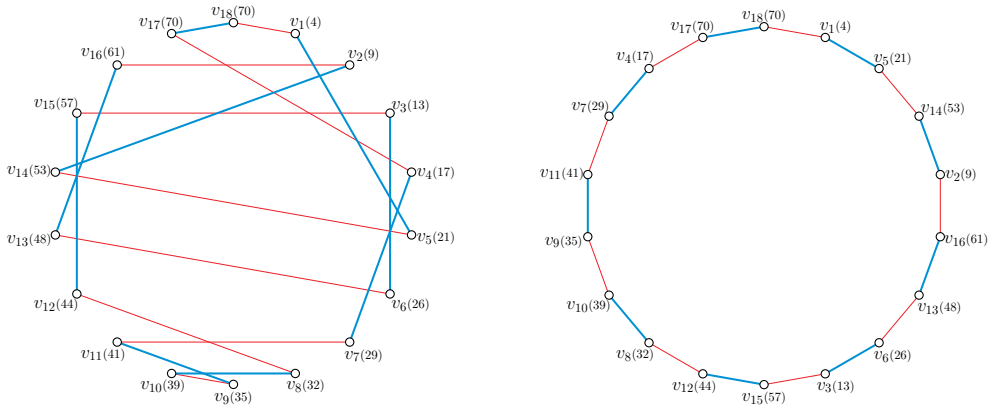


Figure 3.5: (a) A subset of edges from R and the edge set B on G with vertices in non-decreasing order of weight; and (b) by rearranging the vertices, it can be seen that the edges form an alternating Hamiltonian cycle in G, which corresponds to a feasible solution $\mathcal{T}$ for the instance $\mathcal{M}$ of the COP.

vertex of the graph is incident to exactly one edge, and therefore contains $\frac{|V|}{2}$ edges. A perfect matching can only exist in graphs comprising an even number of vertices. Note that the edge set B has cardinality $n+1$; thus B is a perfect matching in G. The solution $\mathcal{T}$ must contain all pairs of values from the problem instance $\mathcal{M}$, thus it follows that all $n+1$ edges in B must be present in the alternating Hamiltonian cycle. Finally, we are now able to formulate our approach to solving the COP.

Corollary 3.6 *Let $G = (V, B \cup R)$ be a simple, undirected graph modelling an instance $\mathcal{M}$ of the COP. Then, finding a solution $\mathcal{T}$ involves finding a perfect matching $R' \subseteq R$ that, together with the perfect matching B, forms an alternating Hamiltonian cycle in G.*

Proof. All edges in B are required in the final cycle as they represent all pairs of values in $\mathcal{M}$, thus if a subset of $n+1$ red edges $R' \subseteq R$ exists that is able to form an alternating Hamiltonian cycle in G with the edges in B it follows directly from Theorem 3.5 that this cycle corresponds to a solution $\mathcal{T}$. □

In a final solution $\mathcal{T}$ for an instance of the COP the outermost values in the sequence are not required to fulfil the vicinal sum constraint, as these values are not adjacent to any other values. This is also true for the outermost score widths in a feasible alignment of items for the sub-SCPP. In this model for the COP, the universal vertices aid the construction of the alternating Hamiltonian cycle as they are able to connect to the vertices that correspond to the two values that will appear in the outermost positions in $\mathcal{T}$. Once an alternating Hamiltonian cycle has been found the universal vertices and any incident edges are removed, resulting in an alternating path corresponding to a feasible sequence $\mathcal{T}$. Figure 3.6 shows the final alternating path

found in the graph G (see Figure 3.4b), representing the solution $\mathcal{T}$. The equivalent sub-SCPP solution using the instance in Figure 3.3 is provided in Figure 3.7.

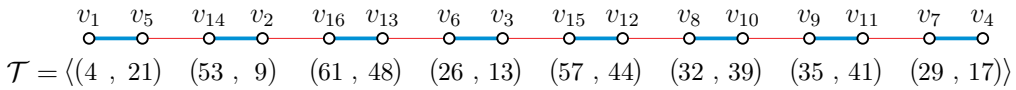


Figure 3.6: The alternating path corresponding to a solution $\mathcal{T}$ obtained by removing the universal vertices.

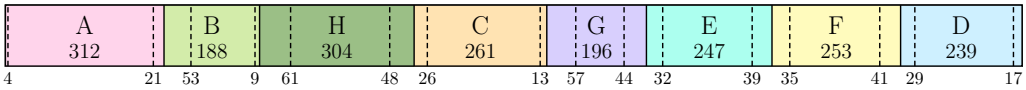


Figure 3.7: The corresponding feasible alignment of items for the equivalent sub-SCPP instance.

Although Goulimis' model and our model both use two vertices to represent each item, the graph in the former model comprises twice as many edges. This is because each vertex has two weights, and so if there exists an edge between two vertices in different vertex subsets (i.e. from two different items) then there also exists an edge between the other two vertices in the vertex subsets in the opposite direction due to symmetry. We also see that in the TPP model only half of the vertices on the graph will be used in a solution. However, in our approach as each vertex is assigned a single weight there is no need for a directed edge set, and so a feasible solution will include every vertex. Moreover, the TPP approach requires all vertices to be indexed in a particular manner, with the regular and rotated orientations of each item being represented by an odd- and even-indexed vertex respectively in lexicographical order. This is no longer necessary in our edge-coloured graph G, as partner vertices from the same pair in $\mathcal{M}$ are connected by a blue edge in the set B, and so we are able to index the vertices in weight order without disrupting the structure of G.

The problem of finding an alternating Hamiltonian cycle in a graph with two edge sets is NP-hard as it is a generalisation of the Hamiltonian cycle problem (Häggkvist, 1977). Despite this, due to the manner in which graphs are modelled from instances of the COP and the requirement that all edges in B must be included, we are able to find an alternating Hamiltonian cycle in a graph G, if one exists, in polynomial-time (Hawa et al., 2018).

3.3 The Alternating Hamiltonian Construction Algorithm

In this section we present the Alternating Hamiltonian Construction (AHC) algorithm, designed to seek alternating Hamitonian cycles within graphs modelling COP instances. As explained, the graph $G = (V, B \cup R)$ comprises two edge sets B and R,

where B is a perfect matching of cardinality $n+1$. The aim is to find a suitable subset R' of edges in R which is able to form an alternating Hamiltonian cycle in G with the edges in B (see Theorem 3.5 and Corollary 3.6). Thus, an alternating Hamiltonian cycle will comprise two perfect matchings, B and R', such that $B \cap R' = \emptyset$.

Firstly, we perform basic preliminary checks to prevent executing AHC unnecessarily. Here, we assess the $2n$ vertices corresponding to the values in the COP instance $\mathcal{M}$. Of these vertices, suppose the two smallest vertices, v_1 and v_2, are placed in the outermost positions in the sequence. It is clear then that if the vertices v_3 and v_{2n} do not fulfil the vicinal sum constraint, i.e. $w(v_3) + w(v_{2n}) < \tau$, there cannot exist a feasible sequence of all the pairs of values in $\mathcal{M}$. Furthermore, if v_1 and v_2 are a pair in $\mathcal{M}$, that is, $p(v_1) = v_2$, and $w(v_2) + w(v_{2n}) < \tau$, it is also not possible to form a feasible ordering of all elements in $\mathcal{M}$. Note, however, that a positive outcome from these tests does not necessarily imply that a feasible solution exists for the given instance, but a negative outcome does confirm the non-existence of a solution.

Once these tests have been completed successfully for a given instance, we can then initiate the Alternating Hamiltonian Construction algorithm. AHC consists of two subprocedures: one to form an initial matching $R' \subseteq R$, and another to alter R' such that it contains suitable edges that form an alternating Hamiltonian cycle in G along with the fixed edges in B.

3.3.1 The Maximum Cardinality Matching Algorithm

The first subprocedure of AHC is the Maximum Cardinality Matching (MCM) algorithm, which is used to find a maximum matching R' from R. Clearly, the aim is to find a maximum matching that is a perfect matching, i.e. with cardinality $n + 1$. Although there are standard matching processes for general graphs such as the Blossom algorithm (Edmonds, 1965), the special structure of G allows us to achieve a matching via a more efficient procedure (Mahadev and Peled, 1994; Becker, 2010).

The pseudocode for MCM is provided in Algorithm 2. MCM considers the vertices $v_1, v_2, \ldots, v_{2n+2}$ in turn, and matches each vertex with the highest-indexed unmatched vertex adjacent in R (Lines 3–8). As the vertices are labelled in weight-ascending order, this results in the lower-weighted vertices being matched with the higher-weighted vertices. The *match* of a vertex v_i is denoted as $m(v_i)$. Then, the set R' contains all edges from R between matched vertices – that is, $R' = \{\{v_i, m(v_i)\} : v_i \in V\}$. Note that there are two cases in which a vertex is not matched with the highest-indexed unmatched vertex. The first occurs when the highest-indexed unmatched vertex is the current vertex v_i's partner, and so v_i must be matched with the next highest-indexed vertex. However, a second scenario arises when v_i is not

Algorithm 2 MCM ($G = (V, B \cup R)$)

```
1:  R' ← ∅
2:  m(v_i) ← NULL ∀ v_i ∈ V
3:  for i ← 1 to 2n + 2 : m(v_i) = NULL do
4:      for j ← 2n + 2 to i + 1 : m(v_j) = NULL do
5:          if {v_i, v_j} ∈ R then
6:              m(v_i) ← v_j, m(v_j) ← v_i
7:              R' ← R' ∪ {{v_i, v_j}}
8:              break
9:      if m(v_i) = NULL and i ≠ 1 and m(v_{i−1}) ≠ NULL
            and m(p(v_i)) = NULL and {v_{i−1}, p(v_i)} ∈ R then
10:         R' ← R'\{{v_{i−1}, m(v_{i−1})}}
11:         m(v_i) ← m(v_{i−1}), m(m(v_i)) ← v_i
12:         m(v_{i−1}) ← p(v_i), m(p(v_i)) ← v_{i−1}
13:         R' ← R' ∪ {{v_{i−1}, p(v_i)}} ∪ {{v_i, m(v_i)}}
14: return R'
```

adjacent to any other unmatched vertex except its partner, $p(v_i)$. In this case, MCM is able to *rematch* v_i using the previous vertex, v_{i-1}, provided:

(i) $i \neq 1$ (the current vertex v_i is not the first vertex);

(ii) $\{v_{i-1}, m(v_{i-1})\} \in R'$ (the previous vertex v_{i-1} has been matched); and

(iii) $\{v_{i-1}, p(v_i)\} \in R$ (the previous vertex v_{i-1} is adjacent to the current vertex's partner, $p(v_i)$).

Then, we simply match v_i with the vertex matched with v_{i-1}, and rematch v_{i-1} with v_i's partner, $p(v_i)$ (Lines 9–13).

The underlying algorithm excludes the rematching procedure and is designed to produce a maximum matching on threshold graphs (Mahadev and Peled, 1995). However, recall that our graph G is a *modified* threshold graph; vertices that meet the vicinal sum constraint can only be connected by an edge in R if they are not partners, even if their combined weight equals or exceeds τ. As the goal is to find a matching $R' \subseteq R$, edges in B cannot be considered for the matching. Therefore, it must be shown that the addition of the rematching procedure to the algorithm is a suitable approach to manage the modification to the graph and produce a maximum cardinality matching, as described in the following theorem attributed to Becker (2010).

Theorem 3.7 *Let $G = (V, B \cup R)$ be a graph modelling an instance $\mathcal{M}$ of the COP. Then, MCM returns a maximum cardinality matching R'.*

Proof. The MCM algorithm is based on the algorithm of Mahadev and Peled (1995) which finds maximum matchings on threshold graphs and does not include the rematching procedure. Throughout MCM, if the current vertex v_i is adjacent to at

least one other unmatched vertex in R, then MCM performs in the same manner as the original algorithm. However, if v_i is not adjacent to any other unmatched vertex except its partner $p(v_i)$, then MCM checks Conditions (i)–(iii) above to determine if a rematching can occur. Clearly, if $i > 1$, a successful rematching increases the cardinality of the matching R'. We now show that rematching is a suitable procedure for producing a maximum cardinality matching R' when excluding edges in B, and that the cardinality of R' cannot be increased in any other way if v_i cannot be rematched and is left unmatched.

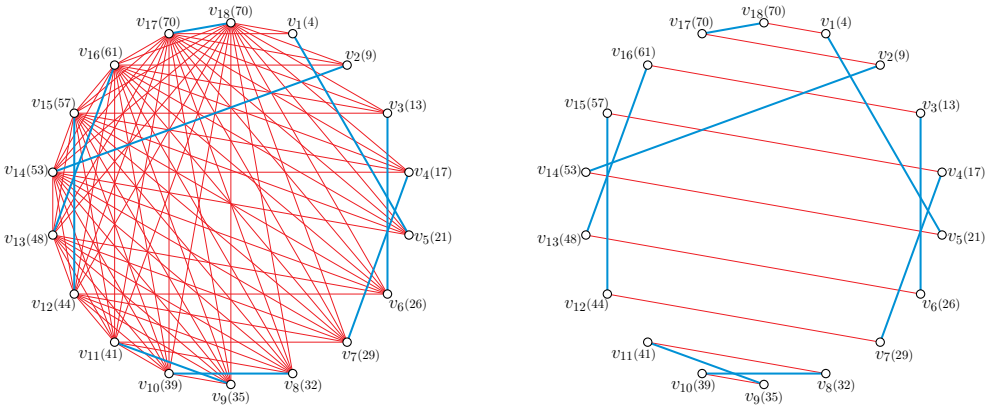
Recall that the vertices on the graph G are labelled in non-decreasing order of weight, and therefore form a non-decreasing degree sequence, i.e. $\deg(v_{i+1}) \geq \deg(v_i)$, although we note that a higher weight does not necessarily imply a higher degree. For example, on the graph G in Figure 3.4b, $w(v_4) = 17$ and $w(v_5) = 21$, so $w(v_5) > w(v_4)$; however $\deg(v_4) = \deg(v_5) = 6$. Now, consider the following observations:

1. When matching a vertex v_i, MCM only has to attempt a rematching with one other vertex because the cardinality of the matching R' is not affected by the particular vertices already matched.
2. It is not possible to rematch v_i using an unmatched vertex v_j of a higher degree as v_i will not be adjacent to any unmatched vertices that are adjacent to v_j except $p(v_i)$.
3. It is not possible to rematch v_i using an unmatched vertex v_j of the same degree as v_i and v_j will be adjacent to the same vertices; thus the only unmatched vertex adjacent to v_j is $p(v_i)$.
4. If v_i is unable to be rematched using vertex v_{i-1}, then it is not possible to rematch v_i using a matched vertex of lower degree than v_{i-1}.
5. If v_i is able to be rematched using vertex v_{i-1}, then it is also possible to rematch v_i using a matched vertex of the same degree as v_{i-1}.
6. If v_{i-1} is not matched then $\deg(v_{i-1}) < \deg(v_i)$, otherwise v_{i-1} would have been matched with $p(v_i)$. Thus, it would not be possible to rematch v_i with any vertices of the same or lower degree than v_{i-1}.

Therefore v_{i-1} is the most suitable vertex for a rematching and there is no other way of improving the cardinality of the matching if a rematching with v_{i-1} is not possible. □

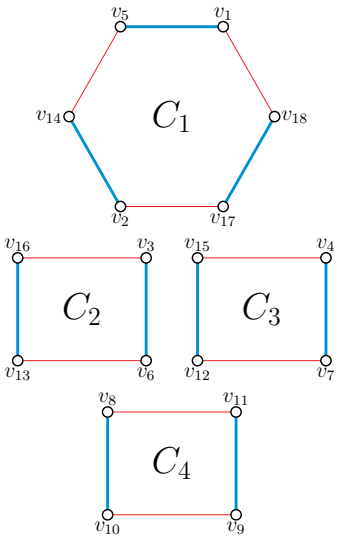
On completion of MCM, the maximum matching R' is evaluated. If R' does not contain $n + 1$ edges, then it is not a perfect matching and there are an insufficient number of edges in R' to form a cycle with the edges in B. Thus, no feasible solution can exist and AHC is terminated. Otherwise, if $|R'| = n + 1$, MCM has successfully

produced a perfect matching. Then, the spanning subgraph $G' = (V, B \cup R')$ is a 2-regular graph consisting of cyclic components $C_1, C_2, \ldots, C_z$, in which every vertex $v_i \in V$ is adjacent to its partner $p(v_i)$ via a blue edge in B and its match $m(v_i)$ via a red edge in R'. Evidently, if G' comprises a single cyclic component (i.e. if $z = 1$) then G' is an alternating Hamiltonian cycle and a solution has been found. On the other hand, if G' is formed of multiple cycles then AHC must find a way of connecting these components together to create a single alternating Hamiltonian cycle. Figure 3.8b shows the MCM process on the instance $\mathcal{M}$ of the COP stated in Section 3.2, showing the red edges of R' connecting the lowest-indexed vertices to the highest-indexed vertices in turn. In this example, no rematching was necessary. The cyclic components of G' are clearly visible by depicting G' in planar form as in Figure 3.8c, where G' comprises $z = 4$ components.



(a) $G = (V, B \cup R)$

(b) $G' = (V, B \cup R')$



(c) G' in planar form

Figure 3.8: (a) The graph G modelling our example instance $\mathcal{M}$; (b) the subgraph $G' = (V, B \cup R')$ with edge set $R' \subseteq R$ produced by MCM; and (c) a planar embedding of G' showing $z = 4$ components.

In order to merge the components together, edges in R' must be replaced with new edges from $R \backslash R'$ that connect vertices from different components. Now, the task involves deciding which particular edges to remove from R' and which edges from $R \backslash R'$ to add to R'.

3.3.2 The Bridge-Cover Recognition Algorithm

Recall that an edge on a graph is a *bridge* if removing the edge increases the number of components of the graph; the addition of a bridge decreases the number of components. To combine the components of G' into a single component AHC calls upon a second subprocedure, the Bridge-Cover Recognition (BCR) algorithm, an iterative procedure based on a method by Becker (2010). BCR is designed to identify subsets of red edges in different components of G' that will be removed from R'. Subsequently, these edge subsets will also be used to identify the new edges from $R \backslash R'$ to be added to R' that will function as bridges, connecting the components of G' into a single component. BCR forms a collection $\mathcal{R}''$ of these edge subsets. The collection $\mathcal{R}''$ is said to *cover* a component C_j of G' if there exists a subset in $\mathcal{R}''$ that contains an edge in C_j. BCR aims to create a collection $\mathcal{R}''$ that covers all z components of G'.

To begin, the edges in R' are sorted into a list $\mathcal{L}$ such that the lower-indexed vertices of the edges are in increasing order and the higher-indexed vertices are in decreasing order. Any edges that did not connect a vertex to the highest-indexed unmatched vertex in MCM cannot be sorted in this manner are removed from the list. An example of this list $\mathcal{L}$ can be seen in Figure 3.9a. During each iteration, BCR searches from the beginning of $\mathcal{L}$ to find two or more successive edges that meet the following conditions:

(i) the lower-indexed vertex of each edge is adjacent to the higher-indexed vertex of the next edge in $\mathcal{L}$;

(ii) each edge is in a different component of G'; and

(iii) only one of the edges is in a component already covered by $\mathcal{R}''$, and all other edges are in components not yet covered by $\mathcal{R}''$.

These edges form a subset R''_i which BCR adds to $\mathcal{R}''$ before continuing through the list in search of another suitable succession of edges.[1] After the penultimate edge in the list has been assessed, edges in $\mathcal{R}''$ are removed from $\mathcal{L}$ and the next iteration begins. Once $\mathcal{R}''$ covers all components of G', BCR ends the search. If no new sets are created during an iteration and $\mathcal{R}''$ does not cover all components, then no more sets

[1] When searching for edges to produce the first subset, R''_1, only Conditions (i) and (ii) are required as $\mathcal{R}'' = \emptyset$.

exist and BCR terminates. Furthermore, if fewer than two edges remain in $\mathcal{L}$ after an iteration, then no more sets can be produced as at least two edges are required to form a new set. In both of these cases, it can be said with absolute certainty that no feasible solution exists for the given instance $\mathcal{M}$ of the COP. Figure 3.9a shows how BCR forms the collection $\mathcal{R}''$ on our example instance, where $\mathcal{R}''$ comprises the subsets $R''_1 = \{\{v_2, v_{17}\}, \{v_3, v_{16}\}, \{v_4, v_{15}\}\}$ and $R''_2 = \{\{v_7, v_{12}\}, \{v_8, v_{11}\}\}$ of successive edges in $\mathcal{L}$ that meet the required conditions. As $\mathcal{R}'' = \{R''_1, R''_2\}$ covers all four components of G', no futher subsets are required.

If a feasible collection $\mathcal{R}''$ covering all z components of G' has been created successfully, then there exists an alternating Hamiltonian cycle in G. The collection $\mathcal{R}''$ contains the exact edges to be removed from R', which in turn indicate the replacement edges from $R \backslash R'$ that form bridges between the components.

To procure the bridges, BCR operates on each subset $R''_i \subset \mathcal{R}''$ as follows: for each edge in R''_i in turn, the edge from $R \backslash R'$ connecting the lower-indexed vertex of the edge to the higher-indexed vertex of the next edge is added to R'. Due to the order of the edges in $\mathcal{L}$, the lower-indexed vertex of the last edge in each subset R''_i will be adjacent to the higher-indexed vertex of the first edge in the subset via an edge in $R \backslash R'$, as the weight of the higher-indexed vertex of the first edge will be greater than or equal to the weight of the higher-indexed vertex of the last edge (see Figure 3.9a). These edges are bridges, connecting vertices from different components, as can be seen in Figure 3.9b. The edges in $\mathcal{R}''$ are then removed from R' so that R' is maintained as a perfect matching of cardinality $n+1$. This modified matching R' is able to form an alternating Hamiltonian cycle in G' with the edge set B. Removing the universal vertices yields an alternating path which corresponds to a feasible solution $\mathcal{T}$ (Figure 3.9d).

In the initial version of the algorithm (Becker, 2010), a procedure is used that searches through $\mathcal{L}$ just once to find edge sets for the collection $\mathcal{R}''$. For some instances, although $\mathcal{R}''$ covers all components of G', the components are unable to be connected into a single alternating Hamiltonian cycle. The issue stems from the requirements for edges to create a set, where in this previous procedure new subsets that are formed can contain edges in multiple components of G' already covered by $\mathcal{R}''$. This causes additional bridges to be added between components that have already been connected, preventing the formation of a single cycle. By introducing Condition (iii) and only permitting *one* edge in a new set R''_i to be in a component that $\mathcal{R}''$ covers, we prevent unnecessary edges being added to G' and ensure that components are linked to produce a single cycle. Figure 3.10 shows the formation of $\mathcal{R}''$ using the initial version of the procedure where the set R''_2 contains edges in *two* components that $\mathcal{R}''$ already covers. Although $\mathcal{R}'' = \{R''_1, R''_2\}$ covers all components

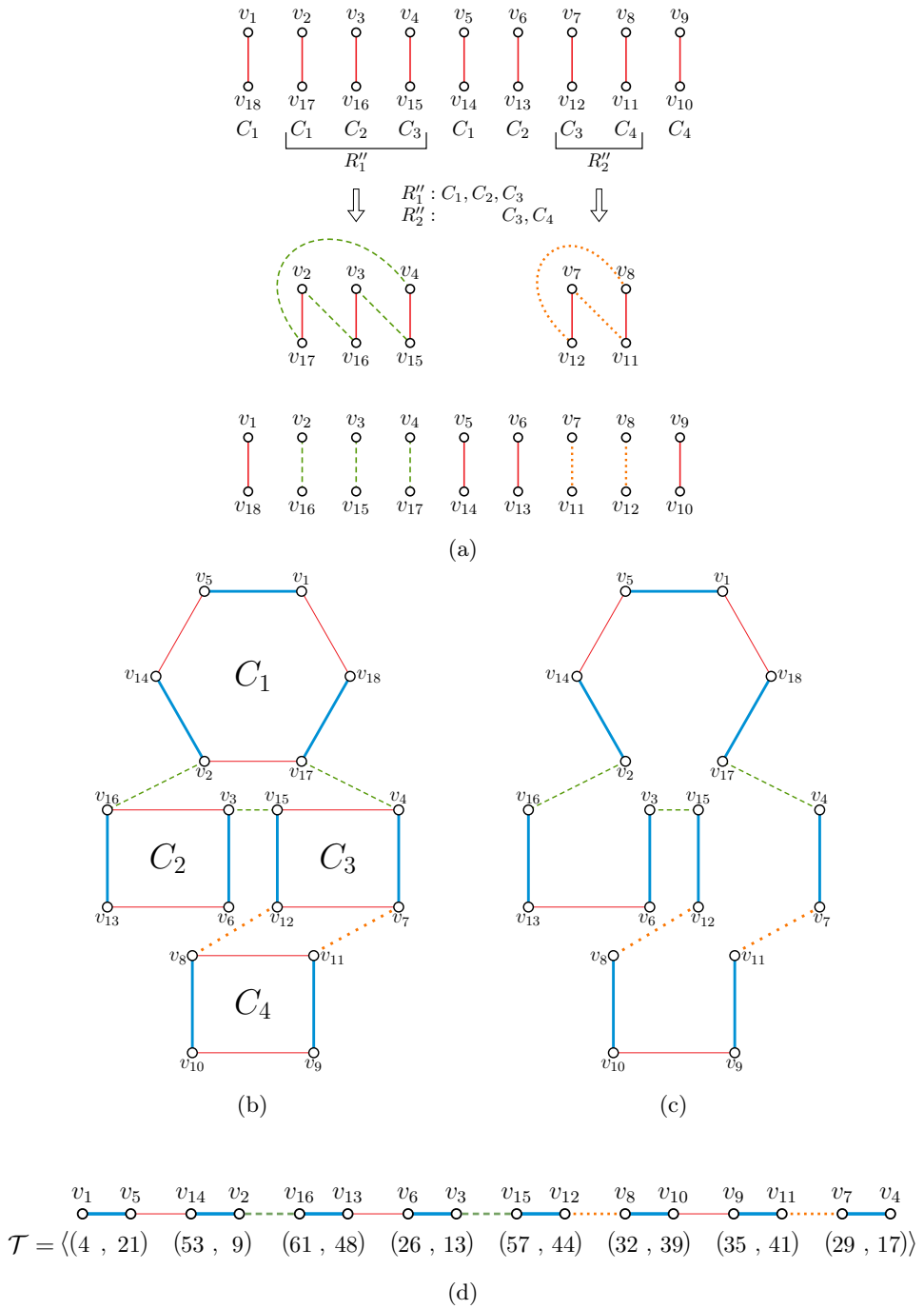


Figure 3.9: BCR creates a collection $\mathcal{R}'' = \{R_1'', R_2''\}$ of edges in R' that when replaced by bridges from $R \backslash R'$ connects the components of G' into a single alternating Hamiltonian cycle. Dashed green edges and dotted orange edges are the bridges from R_1'' and R_2'' respectively. The resulting alternating path corresponds to a solution $\mathcal{T}$.

of G', the bridges obtained from these sets link C_2 and C_3 twice, connecting the four components into two different components. An additional procedure is implemented by Hawa et al. (2018) that recitifies this issue, but we found that the combination of both procedures is unnecessary. Therefore, we replaced the two procedures with a single algorithm, BCR, that produces the same results in a more efficient manner.

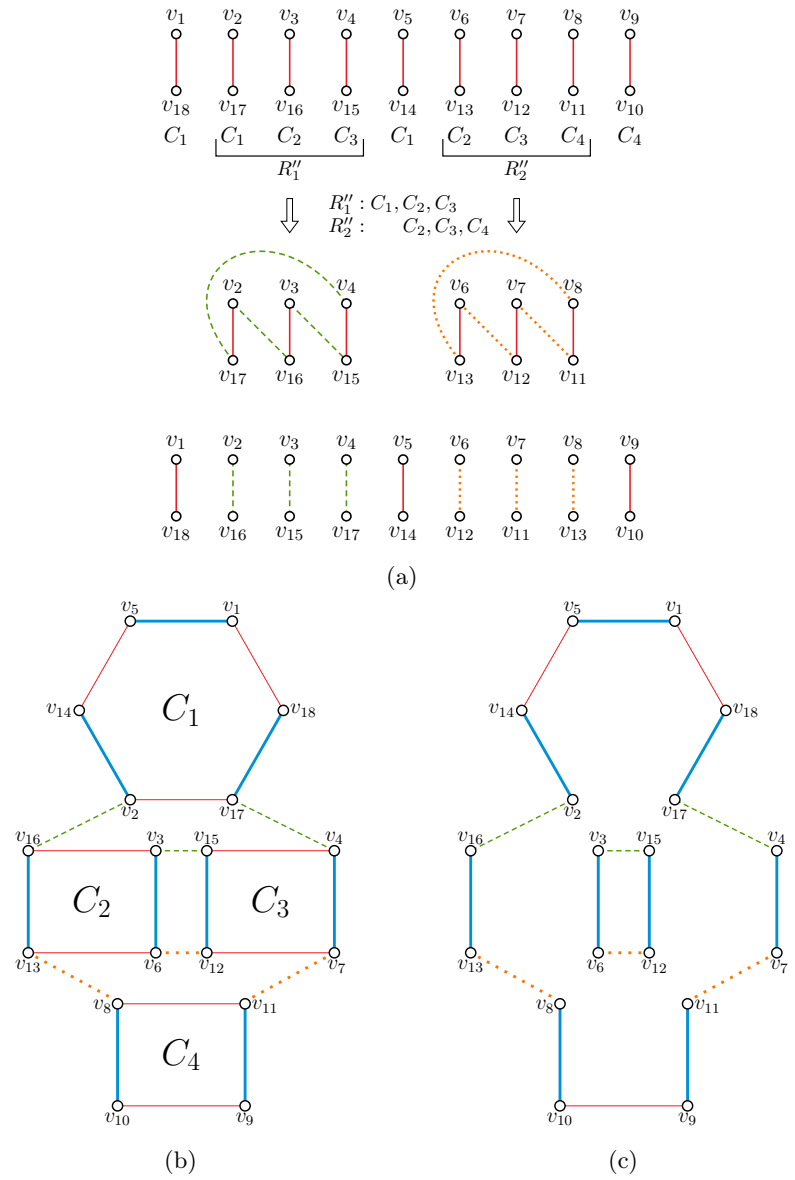


Figure 3.10: The issue caused using the initially proposed procedure to find suitable edge sets, where the collection $\mathcal{R}''$ covers all components of G' but the bridges obtained from the edge sets in $\mathcal{R}''$ form two components, as opposed to a single alternating Hamiltonian cycle.

Theorem 3.8 *Let $G' = (V, B \cup R')$ be a graph consisting of z distinct cyclic components. Then, if BCR produces a collection $\mathcal{R}''$ that covers all components of G' subject to Conditions (i)–(iii), there exists an alternating Hamiltonian cycle in G.*

Proof. As explained previously, a feasible collection $\mathcal{R}''$ leads to an alternating Hamiltonian cycle by removing the selected edges and adding new edges from $R \backslash R'$. We must now show that, without a feasible collection $\mathcal{R}''$, the components of G' cannot be combined into a single cycle.

1. If the adjacency condition does not hold, then there is no edge from $R \backslash R'$ that can connect the vertices between two different components. Hence, specific edges

are removed from the list $\mathcal{L}$ prior to the execution of BCR. This ensures that the adjacency condition holds; that is, for two successive edges (u_1, v_1) and (u_2, v_2) in $\mathcal{L}$, if $(u_1, v_2) \in R \backslash R'$, then $(u_2, v_1) \in R \backslash R'$, due to the manner in which the edges in R' are obtained in MCM.

2. If BCR terminates with a collection $\mathcal{R}'' = \emptyset$, then clearly no suitable edges have been found that can form bridges between any of the components. It follows that no alternating Hamiltonian cycle exists.
3. If BCR terminates with a collection $\mathcal{R}''$ that does not cover all components of G', then the components that are not covered would not be able to be connected to the components that are covered by $\mathcal{R}''$, resulting in multiple components that cannot form an alternating Hamiltonian cycle.
4. If BCR terminates with a collection $\mathcal{R}''$ that covers all components but contains subsets that do not have a common component, then although the individual subsets in $\mathcal{R}''$ will be able to merge components together, it would not be possible to connect the resulting components to one another, and so a single alternating Hamiltonian cycle cannot be formed.

Therefore, it is clear that a feasible collection $\mathcal{R}''$ is required to connect the multiple cyclic components of G' into a single alternating Hamiltonian cycle. □

This concludes the Alternating Hamiltonian Construction (AHC) algorithm. The pseudocode for the entire procedure is provided in Algorithm 3 which returns, for any graph $G = (V, B \cup R)$ formed as described in Section 3.2, an alternating Hamiltonian cycle in G if one exists.

The AHC algorithm is able to determine the existence of a solution and produce a solution if one exists in quadratic time, as stated in the following theorem:

Theorem 3.9 *Let $G = (V, B \cup R)$ be a graph modelled from an instance $\mathcal{M}$ of cardinality n of the COP. Then, AHC terminates in at most $O(n^2)$ time.*

Proof. The first subprocedure, MCM, produces an initial matching $R' \subseteq R$ in at most $O(n^2)$ time. Sorting the $n+1$ edges of R' into a list $\mathcal{L}$ for the second subprocedure, BCR, requires $O(n \log n)$ time. As G' comprises a maximum of $\left\lfloor \frac{n+1}{2} \right\rfloor$ components and each subset R''_i created in BCR must contain at least two edges from R', it follows that the number of subsets in $\mathcal{R}''$ required to cover all components of G' is bounded by $\left\lfloor \frac{n+1}{2} \right\rfloor - 1$. At least one new subset R''_i is created in each iteration of BCR, and removing edges from $\mathcal{L}$ can be performed in constant time, meaning that the task of producing the collection of subsets $\mathcal{R}''$ is of quadratic complexity $O(n^2)$. Up to $n+1$ edges in R' may be replaced with edges from $R \backslash R'$, which can be executed in $O(n)$ time. Consequently, AHC has an overall worst case complexity of $O(n^2)$. □

Algorithm 3 AHC ($G = (V, B \cup R)$)

run preliminary tests on $V \backslash \{v_{2n+1}, v_{2n+2}\}$
if preliminary tests failed **then**
    infeasible, **end**
$R' \leftarrow \text{MCM}(G = (V, B \cup R))$
if $|R'| < n + 1$ **then**
    infeasible, **end**
$G' = (V, B \cup R')$ comprises z cyclic components
if $z = 1$ **then**
    G' is an alternating Hamiltonian cycle
    feasible, **end**
$\mathcal{R}'' \leftarrow \text{BCR}(G' = (V, B \cup R'))$
if $\mathcal{R}''$ covers all z components of G' **then**
    edges in $\mathcal{R}''$ are removed from R' and replaced with edges from $R \backslash R'$
    G' is an alternating Hamiltonian cycle
    feasible, **end**
else infeasible, **end**
return either alternating Hamiltonian cycle G' or statement of infeasibility

3.4 Summary

In this chapter we focused on the Score-Constrained Packing Sub-Problem (sub-SCPP) for packing items into a single bin. After discussing previous methods of modelling the problem we introduced a sequencing problem known as the Constrained Ordering Problem (COP) and explained how any given instance of the sub-SCPP can be simplified as an instance of the COP. We then modelled the COP graphically, showing that the problem is equivalent to a variant of the Hamiltonian cycle problem in which the task is to find a Hamiltonian cycle whereby successive edges alternate between distinct edge sets. From this, we detailed the Alternating Hamiltonian Construction (AHC) algorithm, an exact algorithm that exploits the underlying structure of the graphical model to find a solution to the COP instance in polynomial-time, and thus a solution for the corresponding sub-SCPP instance.

Our AHC algorithm is an improved, more efficient version of a procedure proposed by Becker (2010), which has the ability to provide solutions to problem instances rather than just a simple statement of feasibility or infeasibility, as with Becker's method. Having the actual order and orientation of the items in the feasible alignment is far more beneficial in industry than simply knowing that a solution exists, especially when considering the number of possible configurations of a given set of items. In addition, we also addressed and rectified an issue within the previous procedure that results in invalid solutions being produced for problem instances that in fact possess feasible solutions.

The AHC algorithm is a powerful algorithm which is guaranteed to produce

a feasible solution if one exists, and will prove to be extremely beneficial for the multiple bin problem. Now, given any set of items, we are able to find a feasible alignment of the items and pack them in the correct order and orientation into a bin. Without the AHC algorithm, there is a risk of not being able to identify a feasible configuration of items within a realistic time period. Therefore, it is only natural that we utilise AHC within algorithms and heuristics for the SCPP, as we will discover in subsequent chapters of this thesis.

Chapter 4

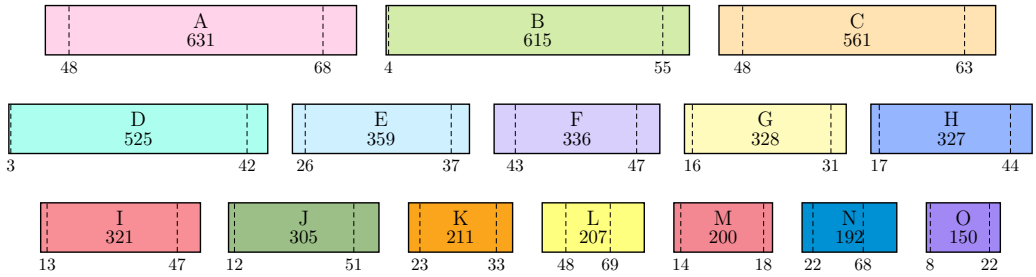
Heuristics for the Score-Constrained Packing Problem

We now turn our attention to the Score-Constrained Packing Problem (SCPP), the multi-bin version of the sub-SCPP where a set of items are to be packed into the fewest number of bins subject to the vicinal sum constraint (see Definition 1.1). In this chapter, we focus on the most straightforward methods for solving the SCPP in the form of constructive heuristics.

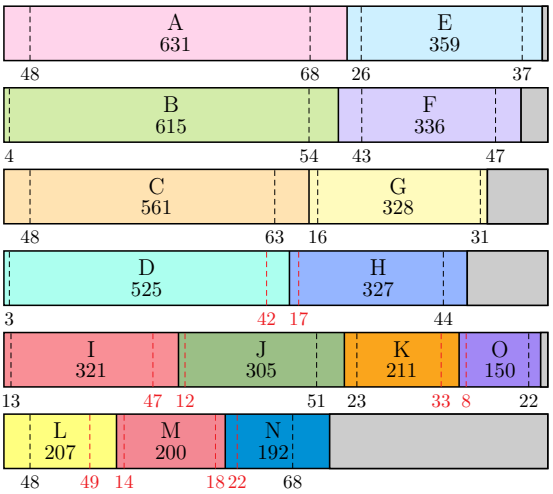
In Chapter 2 it was noted that the SCPP differs from the BPP in that the order and orientation of the items in each bin have a direct effect on the feasibility of the solution due to the vicinal sum constraint (1.1). Therefore, the use of heuristics designed for the BPP to obtain solutions for the SCPP may result in infeasible solutions. This issue is demonstrated in Figure 4.1, where the FFD heuristic for the BPP has been used to produce a solution for an instance $\mathcal{I}$ of the SCPP. The theoretical minimum (2.2) for this instance is $t = 6$, but despite the fact that the resulting solution comprises six bins and is optimal for the BPP (i.e. when $\tau = 0$) the solution is infeasible for the SCPP as the vicinal sum constraint is not satisfied in three of the bins.

Consequently, the author of this thesis developed three heuristics designed specifically for the SCPP: a simple FFD-based heuristic with a minor alteration; a bin-focused heuristic that selects specific score widths; and an advanced adaptation of FFD that integrates the AHC algorithm described in the previous chapter (Hawa et al., 2018). This chapter explores each of these heuristics and their associated characteristics before providing experimental results and comparing the performance of each method.

Recall that an instance $\mathcal{I}$ of the SCPP comprises n items, each with width w_i



(a) An instance $\mathcal{I}$ of the SCPP



(b) An infeasible solution for $\mathcal{I}$ using FFD

Figure 4.1: (a) An instance $\mathcal{I}$ for the SCPP comprising 15 items; and (b) an infeasible solution produced using the FFD heuristic, where the red dashed score lines indicate the vicinal sum constraint violations in three of the bins. In this scenario, $W = 1000$ and $\tau = 70$.

and score widths a_i, b_i for all items $i \in \mathcal{I}$. A solution for an instance $\mathcal{I}$ of the SCPP is represented by $\mathcal{S} = \{S_1, \ldots, S_k\}$ where the bins $S_j \in \mathcal{S}$ have equal capacity W, and $A(S_j)$ denotes the total width of all items in bin S_j. The set $\mathcal{F}$ contains all subsets of items than can be packed into a single bin feasibly.

4.1 The Modified First-Fit Decreasing Heuristic

Our first heuristic for the SCPP is the Modified First-Fit Decreasing (MFFD) heuristic. MFFD acts in the same manner as the original FFD heuristic for the BPP, packing each item in non-increasing order of width into the lowest-indexed bin. However, due to the vicinal sum constraint an additional step is required to ensure feasibility. Once the lowest-indexed bin S_j that can accommodate the current item i has been found, MFFD checks if the vicinal sum constraint is met between the right-most score width in the bin, r_j, and one of the two score widths of i, starting with the smallest score width a_i (Lines 10 and 14 of Algorithm 4). If the constraint is satisfied with one of the score widths, i is packed into S_j in the appropriate orientation, otherwise

MFFD moves on to assess the next bin S_{j+1}. In the event that i cannot be packed into any of the bins in $\mathcal{S}$, MFFD packs i into a new bin in a regular orientation and adds this new bin to the solution $\mathcal{S}$ (Lines 19–23). As with the FFD heuristic for the BPP, the time complexity of MFFD is $O(n \log n)$, because the additional step of checking the score widths meet the vicinal sum constraint can be performed in constant time. Figure 4.2 shows a feasible solution comprising seven bins produced using the MFFD heuristic for the instance $\mathcal{I}$ of the SCPP provided in Figure 4.1a.

Algorithm 4 MFFD($\mathcal{I}$)

```
 1: sort items in I in order of non-increasing widths w_i
 2: let r_j denote the right-most score width in bin S_j
 3: A(S_j) ← 0 ∀ j = 1, ..., |I|                    ▷ All bins are initially empty
 4: S ← ∅
 5: for i ← 1 to |I| do
 6:     j ← 1
 7:     packed ← false
 8:     while not packed do
 9:         if S_j ∈ S and A(S_j) + w_i ≤ W then
10:             if r_j + a_i ≥ τ then
11:                 S_j ← S_j ∪ (a_i, b_i)          ▷ Item i packed in regular orientation
12:                 A(S_j) ← A(S_j) + w_i
13:                 packed ← true
14:             else if r_j + b_i ≥ τ then
15:                 S_j ← S_j ∪ (b_i, a_i)          ▷ Item i packed in rotated orientation
16:                 A(S_j) ← A(S_j) + w_i
17:                 packed ← true
18:             else j ← j + 1
19:         else if S_j ∉ S then
20:             S_j ← S_j ∪ (a_i, b_i)              ▷ Item i packed into a new bin
21:             A(S_j) ← A(S_j) + w_i
22:             S ← S ∪ S_j
23:             packed ← true
24: return S
```

This basic adaptation of the FFD heuristic takes into account the vicinal sum constraint and ensures the feasibility of the solution $\mathcal{S}$. However, we see that MFFD is restricted to only packing items into the ends of the bins. So, although an item i may not be able to be packed into the end of a bin, there could perhaps be another location in the bin that i could be placed into feasibly. Instead, MFFD must pack i into a different bin and may potentially have to open a new bin; thus increasing the number of bins in the final solution. An example of this can be seen in Figure 4.2. Although item K would have been able to fit into bin S_5 (which would have only contained items I and J), neither of the score widths on item K fulfil the vicinal sum constraint with the right-most score width, $r_5 = 12$, and so item K is packed into a new bin. Observe, however, that item K could have in fact been placed *between* items I and J and the vicinal sum constraint would have been satisfied, preventing

the addition of a new bin.

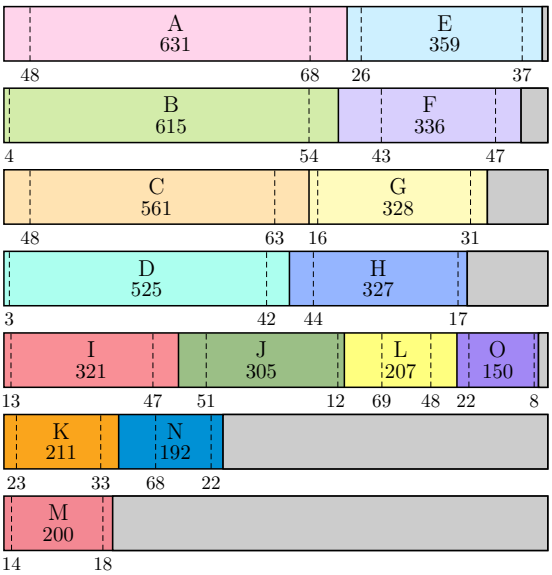


Figure 4.2: A feasible solution comprising $k = 7$ bins created using MFFD for the instance $\mathcal{I}$ in Figure 4.1a, where $W = 1000$, $\tau = 70$, and $t = 6$.

4.2 The Pair-Smallest Heuristic

The Pair-Smallest (PS) heuristic is an extension of an approximate method defined by Lewis et al. (2011). Rather than packing each item in turn, PS focuses on packing each bin individually. To begin, the score widths of all items in $\mathcal{I}$ are put into a list $a_1, \ldots, a_{2|\mathcal{I}|}$ in non-decreasing order. Each item in $\mathcal{I}$ comprises two score widths; thus for each score width a_i we denote the opposite score width on the same item as a_i by $p(a_i)$. Therefore, the item that possesses the score width a_i can be described by its score widths $(a_i, p(a_i))$, with width $w_{(a_i,p(a_i))}$. A bin S_j is initialised by selecting the item from $\mathcal{I}$ with the smallest score width that has not yet been packed and then placing it into S_j in a regular orientation (Lines 17–21 of Algorithm 5). PS then proceeds through the list of score widths to find the smallest score width of an unpacked item that is able to meet the vicinal sum constraint with the right-most score width r_j in the current bin S_j, breaking ties by choosing the score width that has the largest opposite score width (Line 12). If the width of the item does not cause the bin S_j to overfill, then the item is packed into the bin (Lines 13–16); else PS continues on to assess the next score width in the list. Once there are no items suitable for packing into the current bin, either because there are no score widths that can satisfy the vicinal sum constraint with r_j or because all of the remaining unpacked items are unable to fit into the bin, PS closes the current bin and the process is repeated with a new bin (Line 22). As PS searches repeatedly through the

list of score widths, it follows that the worse-case time complexity for PS is $O(n^2)$.

Algorithm 5 PS($\mathcal{I}$)

```
1: sort all score widths a_1, a_2, ..., a_{2|I|} in non-decreasing order
2: let p(a_i) denote the opposite score width on the same item as a_i
3: let r_j denote the right-most score width in bin S_j
4: A(S_j) ← 0 ∀ j = 1, ..., |I|                    ▷ All bins are initially empty
5: S ← ∅
6: I' ← ∅          ▷ Set containing score widths of items that have been packed
7: j ← 1
8: while |I'| < 2|I| do
9:     for i ← 1 to 2|I| do
10:        if a_i ∈ I' then continue          ▷ Item with score width a_i is already packed
11:        if S_j ∈ S then
12:            if r_j + a_i ≥ τ and A(S_j) + w_{(a_i,p(a_i))} ≤ W then
13:                S_j ← S_j ∪ (a_i, p(a_i))
14:                A(S_j) ← A(S_j) + w_{(a_i,p(a_i))}
15:                I' ← I' ∪ {a_i} ∪ {p(a_i)}
16:                i ← 1
17:        else if S_j ∉ S then
18:            S_j ← S_j ∪ (a_i, p(a_i))
19:            A(S_j) ← A(S_j) + w_{(a_i,p(a_i))}
20:            I' ← I' ∪ {a_i} ∪ {p(a_i)}
21:            i ← 1
22:    j ← j + 1
23: return S
```

PS aligns score widths such that their combined width does not greatly exceed τ. This eliminates the possibility of packing larger score widths together unnecessarily, instead reserving the largest score widths for placing alongside the smallest score widths. Unlike MFFD, PS prioritises the vicinal sum constraint, choosing to fulfil this constraint first by searching through the score widths rather than considering the items' widths. A feasible solution created using the PS heuristic for the instance $\mathcal{I}$ of the SCPP in Figure 4.1a is provided in Figure 4.3. Note that in this case the solution produced by PS is an optimal solution, comprising $k = 6$ bins.

4.3 The Modified First-Fit Decreasing with AHC Heuristic

Our final heuristic, the Modified First-Fit Decreasing with AHC (MFFD$^+$), features the Alternating Hamiltonian Construction (AHC) algorithm described in Chapter 3. MFFD$^+$, like MFFD, initially performs in a similar fashion to the FFD heuristic for the BPP by attempting to pack each item $i \in \mathcal{I}$ in non-increasing order of width into the lowest-indexed bin S_j. Again, an extra step is required due to the vicinal sum constraint, however rather than attempting to pack an item i into the end of the bin S_j, MFFD$^+$ calls upon AHC to determine a feasible arrangement of all the

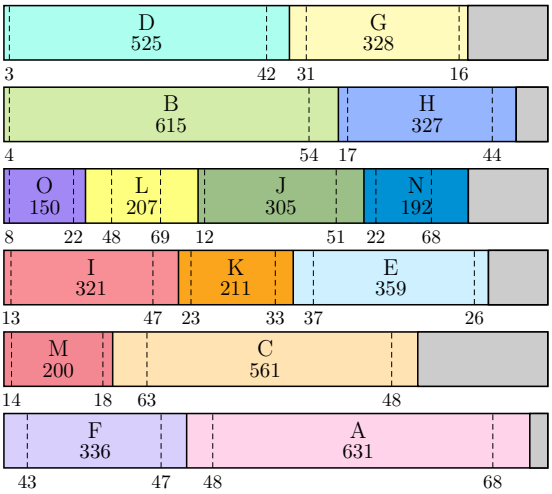


Figure 4.3: A feasible solution produced by PS for the instance $\mathcal{I}$ in Figure 4.1a, where $W = 1000$, $\tau = 70$, and $t = 6$. The solution is optimal as $k = 6$.

items in S_j *and* the current item i (Line 9 of Algorithm 6). If AHC returns a feasible solution the items are placed in the bin in the order of the solution which includes the current item i, otherwise MFFD$^+$ continues on to the next bin (Line 14). As with MFFD, an item i that cannot be placed in any of the bins in $\mathcal{S}$ is packed into a new bin in a regular orientation, which is then inserted into the solution $\mathcal{S}$ (Lines 15–19).

Algorithm 6 MFFD$^+(\mathcal{I})$

```
1: sort items in I in order of non-increasing widths w_i
2: A(S_j) ← 0 ∀ j = 1,...,|I|                    ▷ All bins are initially empty
3: S ← ∅
4: for i ← 1 to |I| do
5:     j ← 1
6:     packed ← false
7:     while not packed do
8:         if S_j ∈ S and A(S_j) + w_i ≤ W then
9:             S' ← AHC(S_j,i)
10:            if S' ∈ F then
11:                S_j ← S'
12:                A(S_j) ← A(S_j) + w_i
13:                packed ← true
14:            else j ← j + 1
15:        else if S_j ∉ S then
16:            S_j ← S_j ∪ (a_i, b_i)            ▷ Item i packed into a new bin
17:            A(S_j) ← A(S_j) + w_i
18:            S ← S ∪ S_j
19:            packed ← true
20: return S
```

Incorporating AHC to solve instances of the sub-SCPP and determine the membership of $\mathcal{F}$ guarantees that a feasible configuration of items will be found if one exists, preventing the unnecessary opening of new bins. Figure 4.4 shows a feasible solution

for the instance in Figure 4.1a produced by MFFD$^+$. Here, in contrast with MFFD, the addition of AHC has allowed a feasible alignment of items I, J, and K in S_5 to be found, which results in fewer bins in the final solution. As with the PS heuristic, MFFD$^+$ has also produced an optimal solution consisting of $k = 6$ bins. Due to the repeated applications of AHC, MFFD$^+$ terminates in at most $O(n^3)$ time.

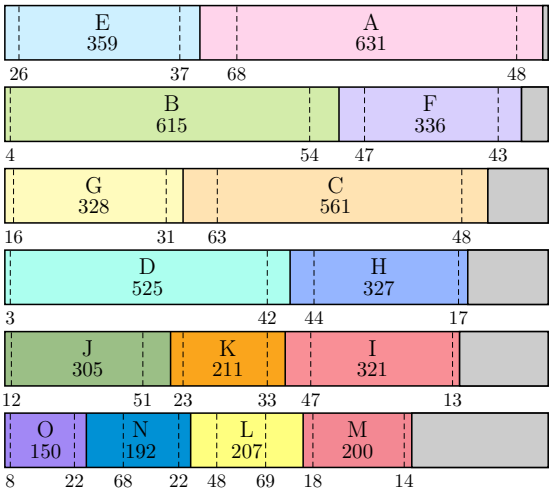


Figure 4.4: A feasible solution obtained using MFFD$^+$ for the instance $\mathcal{I}$ in Figure 4.1a, where $W = 1000$, $\tau = 70$, and $t = 6$. As $k = 6$, this solution is optimal.

4.4 Computational Results

In this section we provide computational results for the three heuristics we have created for the SCPP: the Modified Fist-Fit Decreasing (MFFD) heuristic, the Pair-Smallest (PS) heuristic, and the Modified First-Fit Decreasing with AHC (MFFD$^+$) heuristic. All heuristics were implemented in C++ and executed on Windows machines with Intel Core i5-6500 3.20GHz processors and 8GB of RAM. The results obtained from our experiments can be found online along with all of our source code (Hawa, 2020f).

4.4.1 Problem Instances

Benchmark instances do not exist as of yet for the SCPP, and so for our experiments we used a set of problem instances for the SCPP produced by a problem instance generator developed by the author of this thesis which is available online (Hawa, 2020g). The set contains two types of problem instances: "artificial", in which the items are strongly heterogeneous (the items have varying widths and score widths), and "real", where items are weakly hetergeneous (many items have the same dimensions). Each type contains three subsets of 1000 instances for 100, 500, and 1000 items, giving a total of 6000 problem instances. All items have widths $w_i \in [150, 1000]$ and score

widths $a_i, b_i \in [1, 70]$ selected uniform randomly, and equal height $H = 1$. For the real instances, the number of item types was chosen uniform randomly between 10 and 30, and the number of items within each group also assigned uniform randomly.

Three different bin sizes, $W = 1250$, 2500, and 5000 (also of height $H = 1$) were used in the experiments in order to help alter the number of items per bin. The different combinations of parameters gives rise to 18 different instance classes. We also introduced an additional parameter, δ, which denotes the proportion of pairs of score widths from different items that meet the vicinal sum constraint, i.e. whose combined width equals or exceeds τ. Clearly, when $\delta = 0.0$ none of the items can be packed alongside one another and so the solution will comprise $|\mathcal{I}| = n$ bins (one bin for each item in $\mathcal{I}$), whereas if $\delta = 1.0$ then all possible pairings of score widths will satisfy the constraint and the problem becomes equivalent to the BPP. Here, we determined three values of δ – 0.25, 0.5, and 0.75 – by changing the value of τ.

4.4.2 Analysis of Results

Tables 4.1 to 4.3 contain the results from the experiments using $\delta = 0.25$, 0.5, and 0.75 respectively. All instances were completed in under 262ms. It is worthwhile mentioning that numerous lower bounds exist in the literature for the BPP (see, for example, Martello and Toth (1990b) and Chan et al. (1998)); however, due to the novelty of the SCPP, we have opted for the basic theoretical minimum t (2.2) to compare results from these experiments (see Section 2.8 of Chapter 2). Note that using a different lower bound would not alter the interpretation of the strengths and weaknesses of different algorithms for the SCPP.

A clear pattern can be seen throughout the results. In Table 4.1, when $\delta = 0.25$, the average number of bins $|\mathcal{S}|$ is higher and the difference between $|\mathcal{S}|$ and the theoretical minimum t is large. When moving up to $\delta = 0.75$, in Table 4.3, the average number of bins $|\mathcal{S}|$ is much lower, and so there is a much smaller difference between $|\mathcal{S}|$ and t. This is a result of the changes in δ: when $\delta = 0.25$, a lower proportion of score widths meet the vicinal sum constraint, and so it follows that fewer items can be packed into a single bin together as an increase in items means an increase in the number of adjacent score widths that need to fulfil the vicinal sum constraint. As a result, solutions will tend to consist of many bins containing few items. On the other hand, as δ increases a higher proportion of score widths are able to satisfy the vicinal sum constraint and many items can be packed into each bin feasibly; thus solutions will comprise fewer, well-packed bins overall.

We also observe that the coefficient of variation is considerably higher for real instances than for artificial instances. It is clear that t will be less accurate for real instances due to the lack of item diversity and so, unlike with artificial instances, for

Table 4.1: Results obtained using the MFFD, PS, and MFFD$^+$ heuristics for $\delta = 0.25$. Figures in bold indicate the best results for each instance class. Asterisks indicate statistical significance at $\leq 0.05(^*)$ and $\leq 0.01(^{**})$ according to a two-tailed paired t-test and two-tailed McNemar's test for the $|\mathcal{S}|$ and $\%t$ columns respectively.

$\delta = 0.25$			MFFD		PS		MFFD$^+$	
Type, W	$\lvert\mathcal{I}\rvert$	t^a	$\lvert\mathcal{S}\rvert^b$	$\%t^c$	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$
a, 1250	100	46.13	59.64 ± 5.3	0.0	59.73 ± 5.8	0.0	$^*\mathbf{59.50} \pm 5.4$	0.0
	500	229.37	$\mathbf{289.80} \pm 2.3$	0.0	291.12 ± 2.6	0.0	289.84 ± 2.3	0.0
	1000	458.37	$^{**}\mathbf{573.74} \pm 1.7$	0.0	578.53 ± 1.9	0.0	574.26 ± 1.7	0.0
a, 2500	100	23.32	47.49 ± 7.7	0.0	$^{**}\mathbf{38.91} \pm 9.9$	0.0	46.27 ± 8.6	0.0
	500	114.94	221.92 ± 3.8	0.0	$^{**}\mathbf{186.10} \pm 5.3$	0.0	220.87 ± 4.1	0.0
	1000	229.44	435.27 ± 2.9	0.0	$^{**}\mathbf{374.51} \pm 4.2$	0.0	435.81 ± 3.1	0.0
a, 5000	100	11.92	44.84 ± 8.9	0.0	$^{**}\mathbf{35.43} \pm 12.5$	0.0	42.70 ± 10.4	0.0
	500	57.72	207.22 ± 4.5	0.0	$^{**}\mathbf{168.35} \pm 6.1$	0.0	203.56 ± 4.9	0.0
	1000	114.97	405.41 ± 3.5	0.0	$^{**}\mathbf{335.79} \pm 4.8$	0.0	402.3 ± 3.8	0.0
r, 1250	100	46.44	65.80 ± 13.7	0.0	$\mathbf{65.49} \pm 14.2$	0.0	65.52 ± 14.1	0.0
	500	229.97	324.77 ± 13.1	0.0	$\mathbf{322.97} \pm 13.6$	0.0	332.46 ± 13.4	0.0
	1000	459.38	649.09 ± 13.1	0.0	$\mathbf{645.41} \pm 13.6$	0.0	646.49 ± 13.4	0.0
r, 2500	100	23.47	53.87 ± 17.9	0.0	$^{**}\mathbf{45.67} \pm 22.3$	0.0	51.93 ± 19.6	0.0
	500	115.24	267.71 ± 16.7	0.0	$^{**}\mathbf{223.45} \pm 21.3$	0.0	257.99 ± 18.4	0.0
	1000	229.95	535.67 ± 16.7	0.0	$^{**}\mathbf{446.36} \pm 21.3$	0.0	516.00 ± 18.4	0.0
r, 5000	100	11.98	51.96 ± 20.2	0.0	$^{**}\mathbf{42.35} \pm 27.5$	0.0	49.35 ± 22.3	0.0
	500	57.87	259.80 ± 18.6	0.0	$^{**}\mathbf{207.45} \pm 26.2$	0.0	246.96 ± 20.7	0.0
	1000	115.23	519.95 ± 18.6	0.0	$^{**}\mathbf{414.52} \pm 26.1$	0.0	494.44 ± 20.7	0.0

[a] $t = \lceil \sum_{i=1}^{n} w_i / W \rceil$ (mean from 1000 instances).
[b] Number of bins per solution (mean from 1000 instances plus or minus the coefficient of variation (%)).
[c] Percentage of instances in which the solution comprises t bins.

each individual real problem instance the number of bins in the final solution may differ drastically from one another, resulting in a larger variation.

Looking at Table 4.1 when $\delta = 0.25$, PS produces solutions using fewer bins on average for 15 of the 18 instance classes. In 12 of these 15 cases, namely the six instance classes from both artificial and real instance types when $W = 2500$ and $W = 5000$, these differences are statistically significant. This is to be expected of PS, which prioritises the vicinal sum constraint. As $\delta = 0.25$, fewer score widths from different items will be able to be paired together and meet the constraint. A use of PS ensures that the constraint will be fulfilled, else a new bin will be opened. This is in contrast with MFFD and MFFD$^+$ where the items must be packed in the order they are given. As a result of the small value of δ it can be seen that none of the heuristics are able to produce a solution comprising t bins in any of the instance classes, however it may be that some of the solutions produced are in fact optimal.

Moving on to Table 4.2, when $\delta = 0.5$, PS is still generating solutions using the fewest bins on average for both artificial and real instances when $W = 2500$ and $W = 5000$. However, MFFD$^+$ now produces the best solutions on average for both artificial and real instance types when $W = 1250$ (i.e. when the average number of items per bin is smallest), and these differences are also statistically significant. In

Table 4.2: Results obtained using the MFFD, PS, and MFFD$^+$ heuristics for $\delta = 0.5$. Figures in bold and asterisks should be interpreted as in Table 4.1.

$\delta = 0.5$			MFFD		PS		MFFD$^+$	
Type, W	$\lvert\mathcal{I}\rvert$	t	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$
a, 1250	100	46.13	50.04 ± 5.4	0.3	53.80 ± 6.6	0.0	$^{**}\mathbf{49.73} \pm 5.7$	**0.8**
	500	229.37	240.40 ± 2.2	0.0	262.69 ± 3.0	0.0	$^{**}\mathbf{239.15} \pm 2.3$	0.0
	1000	458.37	474.60 ± 1.5	0.0	520.38 ± 2.2	0.0	$^{**}\mathbf{472.35} \pm 1.6$	0.0
a, 2500	100	23.32	30.75 ± 9.2	0.0	$^{**}\mathbf{24.32} \pm 5.0$	****12.3**	28.46 ± 10.4	2.6
	500	114.94	140.21 ± 4.6	0.0	$^{**}\mathbf{118.52} \pm 2.0$	0.0	132.65 ± 4.9	0.0
	1000	229.44	271.92 ± 3.3	0.0	$^{**}\mathbf{236.00} \pm 1.4$	0.0	258.39 ± 3.5	0.0
a, 5000	100	11.92	23.58 ± 14.8	0.0	$^{**}\mathbf{12.87} \pm 14.7$	****61.2**	19.88 ± 18.3	0.7
	500	57.72	103.21 ± 7.7	0.0	$^{**}\mathbf{58.66} \pm 2.6$	****21.8**	89.54 ± 9.3	0.0
	1000	114.97	198.33 ± 5.8	0.0	$^{**}\mathbf{116.51} \pm 1.8$	0.0	172.61 ± 7.1	0.0
r, 1250	100	46.44	56.76 ± 15.5	0.0	58.12 ± 16.1	0.0	$^{**}\mathbf{56.51} \pm 15.9$	0.0
	500	229.97	279.55 ± 14.6	0.0	287.10 ± 15.2	0.0	$^{**}\mathbf{278.39} \pm 14.9$	0.0
	1000	459.38	558.71 ± 14.6	0.0	573.64 ± 15.1	0.0	$^{**}\mathbf{556.37} \pm 14.8$	0.0
r, 2500	100	23.47	37.07 ± 21.8	0.5	$^{**}\mathbf{28.19} \pm 22.1$	**2.6**	35.42 ± 23.1	1.6
	500	115.24	184.11 ± 20.2	0.0	$^{**}\mathbf{136.35} \pm 20.5$	0.0	177.25 ± 21.2	0.0
	1000	229.95	368.45 ± 20.2	0.0	$^{**}\mathbf{272.19} \pm 20.4$	0.0	355.04 ± 21.2	0.0
r, 5000	100	11.98	32.35 ± 29.4	0.1	$^{**}\mathbf{19.04} \pm 45.6$	****24.0**	29.61 ± 32.7	0.5
	500	57.87	163.82 ± 26.5	0.0	$^{**}\mathbf{89.43} \pm 44.8$	****0.9**	153.42 ± 28.9	0.0
	1000	115.23	328.62 ± 26.4	0.0	$^{**}\mathbf{178.27} \pm 44.8$	**0.1**	308.64 ± 28.7	0.0

fact, PS produces solutions that use the most bins on average of the three heuristics for these instance classes. It can also be seen that, as a larger proportion of score widths can be placed alongside one another, the heuristics are able to find solutions comprising t bins in some cases.

Finally, increasing δ to 0.75 (see Table 4.3), we see that not only does MFFD$^+$ produce solutions using the fewest number of bins on average for artificial and real instance classes when $W = 1250$ (as with $\delta = 0.5$) but also for the three artificial instance classes when $W = 2500$, as well as the artificial instance classes when $|\mathcal{I}| = 1000$ and $W = 5000$. In addition, although PS produces the best solutions on average for artificial instances with $|\mathcal{I}| = 500$ and $W = 5000$, as well as real instances with $|\mathcal{I}| = 100$ and $W = 2500$, MFFD$^+$ produces the most solutions comprising t bins in both cases, with the difference being statistically significant. Furthermore, we see that when $|\mathcal{I}| = 1000$ and $W = 5000$ for both artificial and real instance classes, both MFFD and MFFD$^+$ are able to find solutions comprising t bins; however PS is unable to find a single solution containing t bins.

This clear trend can be attributed to the mechanisms of the heuristics. When δ is smaller, fewer score widths can be aligned feasibily next to one another, and so focusing on selecting items whose score widths will meet the vicinal sum constraint, as with PS, will produce better solutions in comparison to the other methods which are not capable of selecting specific items and must pack the items in the order they are given.

However, the larger value of δ allows for a higher proportion of score widths to be

Table 4.3: Results obtained using the MFFD, PS, and MFFD$^+$ heuristics for $\delta = 0.75$. Figures in bold and asterisks should be interpreted as in Table 4.1.

$\delta = 0.75$			MFFD		PS		MFFD$^+$	
Type, W	$\lvert\mathcal{I}\rvert$	t	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$
a, 1250	100	46.13	47.87 ± 6.0	8.0	52.28 ± 6.8	0.0	$^{**}\mathbf{47.76} \pm 6.1$	$^{**}\mathbf{10.4}$
	500	229.37	232.46 ± 2.1	0.0	251.32 ± 3.1	0.0	$^{**}\mathbf{232.04} \pm 2.1$	0.0
	1000	458.37	462.56 ± 1.4	0.0	496.42 ± 2.3	0.0	$^{**}\mathbf{462.06} \pm 1.4$	0.0
a, 2500	100	23.32	24.66 ± 6.1	24.3	24.13 ± 4.8	21.3	$^{**}\mathbf{23.94} \pm 5.5$	$^{**}\mathbf{55.7}$
	500	114.94	117.74 ± 2.4	3.2	118.03 ± 2.0	0.0	$^{**}\mathbf{116.09} \pm 2.1$	$^{**}\mathbf{21.3}$
	1000	229.44	233.24 ± 1.6	0.0	235.13 ± 1.4	0.0	$^{**}\mathbf{231.16} \pm 1.4$	0.0
a, 5000	100	11.92	14.14 ± 11.9	12.2	$^{**}\mathbf{12.08} \pm 4.9$	$^{**}\mathbf{83.8}$	12.48 ± 8.7	63.6
	500	57.72	62.88 ± 4.8	1.2	$^{**}\mathbf{58.43} \pm 2.0$	29.4	58.56 ± 2.7	$^{**}\mathbf{50.9}$
	1000	114.97	122.16 ± 3.2	0.3	116.32 ± 1.4	0.0	$^{**}\mathbf{115.91} \pm 1.6$	$^{**}\mathbf{37.8}$
r, 1250	100	46.44	53.50 ± 17.0	0.0	56.69 ± 16.4	0.0	$^{**}\mathbf{53.39} \pm 17.2$	0.0
	500	229.97	263.42 ± 16.1	0.0	280.24 ± 15.4	0.0	$^{**}\mathbf{263.03} \pm 16.2$	0.0
	1000	459.38	526.32 ± 16.0	0.0	559.99 ± 15.4	0.0	$^{**}\mathbf{525.60} \pm 16.1$	0.0
r, 2500	100	23.47	28.34 ± 18.5	13.4	$^{**}\mathbf{25.08} \pm 13.0$	4.7	27.69 ± 18.4	$^{**}\mathbf{17.9}$
	500	115.24	140.51 ± 17.2	0.0	$^{**}\mathbf{123.36} \pm 12.5$	0.0	138.56 ± 17.0	0.0
	1000	229.95	281.23 ± 17.2	0.0	$^{**}\mathbf{246.29} \pm 12.5$	0.0	277.84 ± 17.1	0.0
r, 5000	100	11.98	19.91 ± 33.7	9.9	$^{**}\mathbf{12.56} \pm 17.5$	$^{**}\mathbf{68.0}$	18.28 ± 34.8	19.2
	500	57.87	100.87 ± 31.1	1.3	$^{**}\mathbf{60.39} \pm 16.2$	**3.0**	96.73 ± 31.5	1.9
	1000	115.23	202.55 ± 31.1	**0.3**	$^{**}\mathbf{120.27} \pm 16.3$	0.0	195.60 ± 31.4	**0.3**

placed alongside one another legally; a method such as PS is not always necessary as for each score width r_j on the end of a bin S_j there will be a greater number of score widths from unpacked items that can fulfil the vicinal sum constraint. PS selects the unpacked item with the smallest score width that meets the constraint with r_j without considering the effect of the item's width on the packing, which could then cause the number of bins in the final solution to increase. Instead, especially for smaller bin sizes, MFFD$^+$ excels because the large value of δ means that the problem instances are closer to instances of the BPP, for which we have seen FFD performs well.

4.5 Summary

In this chapter we embarked on our first and simplest approach to the SCPP in the form of constructive heuristics. We presented three heuristic methods explicitly crafted for the SCPP: the Modified First-Fit Decreasing (MFFD) heuristic, a basic upgrade from the FFD algorithm for the BPP; the Pair-Smallest (PS) heuristic, which packs individual bins in turn; and the Modified First-Fit Decreasing with AHC (MFFD$^+$) heuristic, which features the AHC algorithm from the previous chapter to solve instances of the sub-SCPP. Results obtained from thorough experiments using a variety of problem instances classes, bin sizes, and proportions of score widths that meet the vicinal sum constraint were then provided and reviewed, clearly showing the different types of instances for which specific heuristics obtain superior solutions.

The obvious disadvantage to these heuristic methods is the inability to move

items once they have been packed into a bin. Although in MFFD$^+$ the items can be rearranged within each bin, none of the heuristics permit the movement of items *between* bins. Essentially, these heuristics comprise a single iteration with no possibility of further amendment. It is this downside that leads us to explore further methods that allow solutions to be adapted and modified.

Chapter 5

Evolutionary Methods for the Score-Constrained Packing Problem

Recall from Chapter 2 that an evolutionary algorithm (EA) is a population-based metaheuristic that emulates natural evolution. In general, EAs implement processes inspired by reproduction, recombination, and mutation to iteratively generate new solutions from existing candidate solutions, helping to evolve and improve the population. Using a fitness function to evaluate the quality of the solutions with respect to the problem at hand, EAs aim to preserve higher quality solutions in the population whilst eliminating lower quality solutions throughout the generations.

EAs have previously been implemented for a variety of grouping problems such as vehicle routing (Pankratz, 2005), graph colouring (Lewis, 2015a) and timetabling and scheduling (Falkenauer and Bouffouix, 1991; Lewis and Paechter, 2007), as well as cutting and packing problems (Reeves, 1996; Hinterding and Khan, 1994; Burke et al., 2006; Junkermeier, 2015; Lewis and Holborn, 2017). As with the BPP, EAs are a suitable approach for the SCPP since it is NP-hard; the set of all possible feasible solutions for any given non-trivial problem instance will usually be too large to completely enumerate.

In this chapter we introduce an EA framework for the SCPP and explain the various stages of the algorithm, which include a local search method and three alternative recombination operators. The AHC algorithm presented in Chapter 3 is also integrated to solve instances of the sub-SCPP that arise within the EA. Finally, we provide results from extensive testing and analyse the effect of each recombination operator on different classes of problem instances.

5.1 Representation

Within EAs, candidate solutions to the optimisation problem at hand are often represented or "encoded" using strings of characters or binary values referred to as *chromosomes*, corresponding to the terminology used in natural genetics. The *genes* of each chromosome relate to the individual components of the solution.

For grouping problems, the most straightforward encoding involves assigning one gene per element. For example, the chromosome (2,1,3,1,2,4) would represent the solution where the second and fourth elements are in group 1, the first and fifth elements are in group 2, the third element is in group 3, and the sixth element is in group 4. The main problem, however, with this type of solution encoding is that it contradicts the Principle of Minimal Redundancy (Radcliffe et al., 1991), which states that each member of the search space (i.e. the set of all possible feasible solutions) should be represented by the fewest number of distinct chromosomes so as to reduce the overall size of the search space. Observe that the solution represented by the above chromosome can also be encoded using another distinct chromosome by simply relabelling the groups, for instance (1,3,4,3,2,1). In fact, a candidate solution comprising k groups can be represented by $k!$ different chromosomes using this encoding scheme.

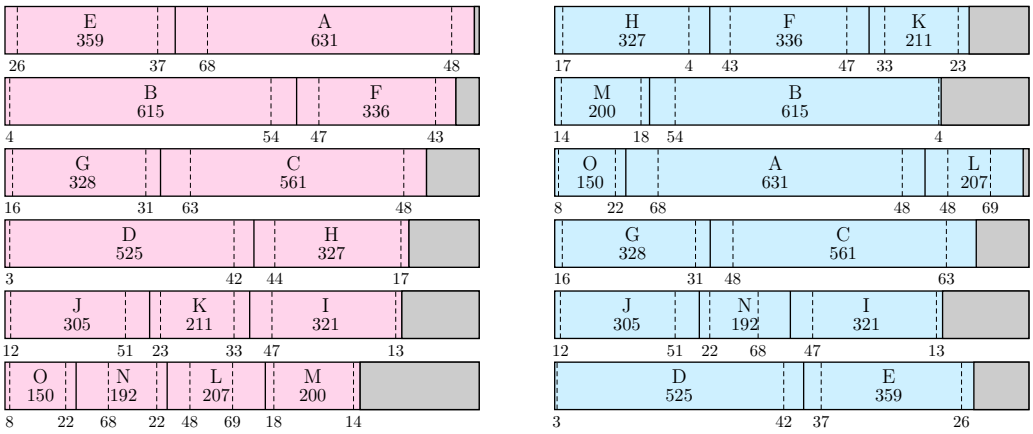
Falkenauer (1993), recognising the issues with standard representations for grouping problems, presented a novel encoding scheme where the standard encoding is combined with a group-based encoding part to create a full chromosome. However, for the SCPP it is also necessary to know the ordering and orientation of the individual items within the bins of a solution. Therefore, in our EA framework we continue to use the description of a solution $\mathcal{S}$ for an instance $\mathcal{I}$ of the SCPP as stated in Section 2.8 of Chapter 2.

5.2 Recombination

Recombination, also referred to as "crossover", is used in EAs to generate new solutions by taking existing *parent* solutions and combining parts of them to yield one or more *offspring* solutions. The recombination operator determines which elements from each parent should be inherited by the offspring with the aim of retaining the best characteristics of each parent such that superior offspring are produced. Examples of basic recombination operators include the single-point and two-point crossovers where groups of genes between randomly selected positions on two parents chromosomes are swapped to create two new offspring, and uniform crossover where genes are chosen from two parent chromosomes with equal probability to produce two offspring (Syswerda, 1989). These classical operators are typically used on binary,

integer, and string encodings.

Note, however, that these operators often result in infeasible offspring for grouping problems. Consider the two parents solutions $\mathcal{S}_1$ and $\mathcal{S}_2$ in Figure 5.1 for an instance $\mathcal{I}$ of the SCPP. Swapping the two central bins between the parent solutions would result in two offspring solutions that not only contain duplicate items, but will also be missing items from the instance $\mathcal{I}$.



(a) $\mathcal{S}_1$ (b) $\mathcal{S}_2$

Figure 5.1: Parent solutions $\mathcal{S}_1$ and $\mathcal{S}_2$ from a population of solutions for the instance $\mathcal{I}$ of the SCPP shown in Figure 4.1a, where $|\mathcal{I}| = 15$, $W = 1000$, and $\tau = 70$.

Furthermore, for the SCPP it is preferable to copy entire bins from parents to offspring rather than individual items. The reason for this is two-fold: firstly, the total width of the new subset of items may exceed the bin capacity; and secondly, even if the items do not cause the bin to overfill there may not exist a feasible configuration of the items such that the vicinal sum constraint (1.1) is satisfied. An example of this scenario is depicted in Figure 5.2, where the location of the items C, E, H, and O have been swapped between the parent solutions to produce two new offspring – that is, the items have been removed from their respective bins in the parent solutions and moved into the same bin in the other parent solution. Each offspring solution contains an overfilled bin and a bin where the new adjacent score widths do not fulfil the vicinal sum constraint; thus both offspring are infeasible. Observe that in this case, no feasible alignments of the items exist for either of the two bins that do not meet the vicinal sum constraint. Note also that this method is only applicable to solutions containing an equal number of bins. Consequently, standard recombination operators are not suitable for the SCPP and alternative operators for our EA must be sought to obtain feasible solutions.

In this section we present three different recombination operators which start with a single empty offspring solution $\mathcal{S} = \emptyset$ and employ two distinct parent solutions, $\mathcal{S}_1$ and $\mathcal{S}_2$, to build the full offspring solution. These recombination operators have

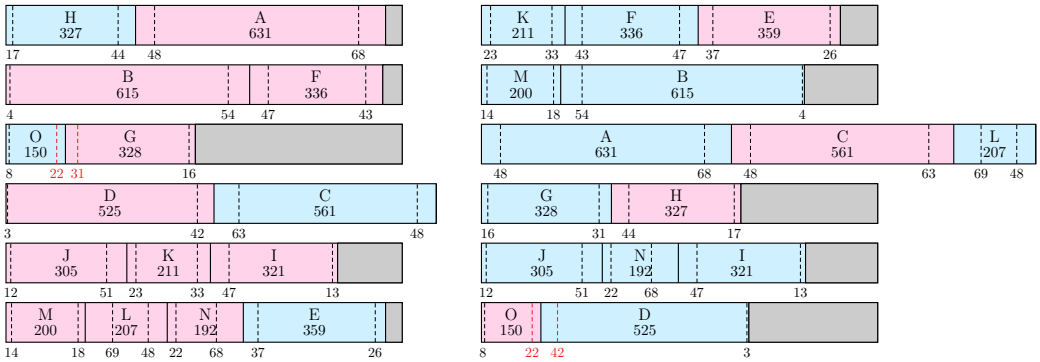


Figure 5.2: Two infeasible offspring solutions produced by swapping the locations of items C, E, H, and O between the parent solutions $\mathcal{S}_1$ and $\mathcal{S}_2$ in the previous figure, where both offspring comprise bins which are overfilled and that violate the vicinal sum constraint.

been designed specifically for the SCPP in that the vicinal sum constraint and the bin capacity constraint will always be satisfied, ensuring all bins in the offspring are feasible; however this may result in a partial offspring solution. In such cases, a repair operator (described below) is called upon to re-establish a full solution $\mathcal{S}$. Examples of how each operator performs will be demonstrated using the parent solutions $\mathcal{S}_1$ and $\mathcal{S}_2$ shown in Figure 5.1. Note that the parent solution $\mathcal{S}_1$ and $\mathcal{S}_2$ used in each of the following three recombination operators are in fact copies, so that the original parent solutions in the population are not modified.

5.2.1 The Grouping Genetic Algorithm Crossover

Our first recombination operator is based on the grouping genetic algorithm of Falkenauer and Delchambre (1992) which, as the name suggests, is designed for grouping and partitioning problems. The pseudocode for this operator, abbreviated to GGA, is provided in Algorithm 7. To begin, the bins of the second parent solution $\mathcal{S}_2$ are randomly permuted and two bins S_x and S_y from $\mathcal{S}_2$ are randomly selected (where $1 \leq x < y \leq |\mathcal{S}_2|$). All bins between and including S_x and S_y are then copied into the offspring solution $\mathcal{S}$ (Line 4). Then, GGA searches through the first parent solution $\mathcal{S}_1$ to find bins that do not contain any items that are already in $\mathcal{S}$ (Lines 5–10). If such a bin is found, GGA adds this bin to $\mathcal{S}$ (Lines 11–12). Note that the bins selected from $\mathcal{S}_2$ cannot be the outermost bins – that is, GGA cannot choose both $x = 1$ *and* $y = |\mathcal{S}_2|$ as doing so would result in the offspring $\mathcal{S}$ inheriting all bins from $\mathcal{S}_2$, preventing the opportunity for $\mathcal{S}$ to inherit bins from $\mathcal{S}_1$.[1]

Figure 5.3 shows the resulting partial offspring solution $\mathcal{S}$ produced using GGA on the parent solutions $\mathcal{S}_1$ and $\mathcal{S}_2$ from Figure 5.1. Here, $x = 2$ and $y = 4$ have been

[1] Although GGA cannot select the outermost bins in $\mathcal{S}_2$ there is still no guarantee that bins from $\mathcal{S}_1$ will be added to the offspring $\mathcal{S}$, as it may be that every bin in $\mathcal{S}_1$ contains an item that is in $\mathcal{S}$.

Algorithm 7 GGA ($\mathcal{S}_1$, $\mathcal{S}_2$)

```
1: S ← ∅
2: permute the bins of parent S_2
3: randomly select bins S_x and S_y from S_2 such that 1 ≤ x < y ≤ |S_2|
4: S ← S ∪ {S_x} ∪ ··· ∪ {S_y}
5: for j ← 1 to |S_1| do
6:     duplicate ← false
7:     for i ← 1 to |S_j| do
8:         if i is in a bin in S then
9:             duplicate ← true
10:            break
11:    if not duplicate then
12:        S ← S ∪ {S_j}
13: return S
```

chosen, so bins S_2, S_3 and S_4 are added to $\mathcal{S}$ from $\mathcal{S}_2$. GGA then adds bins S_4 and S_5 from $\mathcal{S}_1$ to $\mathcal{S}$ as they are the only bins that do not contain duplicate items. On completion, there are three items that have not been packed into $\mathcal{S}$.

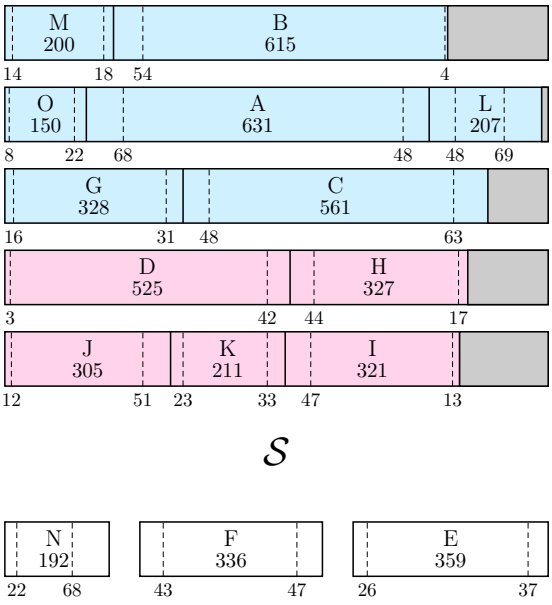


Figure 5.3: The partial offspring $\mathcal{S}$ created using the GGA recombination operator, where bins S_2, S_3, and S_4 have been copied from parent $\mathcal{S}_2$ and bins S_4 and S_5 have been copied from parent $\mathcal{S}_1$. As a result, three items remain unpacked.

The main goal of GGA is to transfer promising bins from parent solutions to offspring. Thus, if a bin S is consistently superior to other bins (in that solutions in the population containing S are of consistently higher quality than solutions that do not contain S), GGA will aid the promotion and transfer of S throughout the population. Therefore, the probability of selecting a parent solution containing S for recombination will increase with each iteration of the EA.

5.2.2 The Alternating Grouping Crossover Using Bin Fullness

The next recombination operator is the alternating grouping crossover using bin fullness (AGX), which is comparable to that proposed for the one-dimensional BPP by Quiroz-Castellanos et al. (2015). Starting with the parent solution containing the fullest bin (breaking ties randomly), AGX transfers this fullest bin into the offspring solution $\mathcal{S}$ (Line 6 of Algorithm 8). Any bins containing items in $\mathcal{S}$ are then removed from the other parent solution (Lines 7–13). AGX then inserts the fullest bin from this modified parent solution into $\mathcal{S}$ and removes bins from the first parent solution. This process continues, alternating between parent solutions and selecting the fullest bin $\max_{S_j \in \mathcal{S}_k}(A(S_j))$ (where $\mathcal{S}_k$ is the parent solution under consideration, $k \in \{1, 2\}$) until at most $\min(|\mathcal{S}_1|, |\mathcal{S}_2|) - 1$ bins (calculated using the initial unmodified parent solutions) have been added to the offspring solution, or the parent solution under consideration is empty (Line 5).

Algorithm 8 AGX ($\mathcal{S}_1$, $\mathcal{S}_2$)

```
1:  S ← ∅
2:  k ← 0
3:  q ← min(|S_1|, |S_2|) − 1
4:  let S_k be the parent solution containing the fullest bin
5:  while |S| < q or S_k = ∅ do
6:      S ← S ∪ max_{S∈S_k}(A(S))          ▷ Add the fullest bin in S_k to S
7:      if k = 1 then k ← 2
8:      else if k = 2 then k ← 1
9:      for j ← 1 to |S_k| do
10:         for i ← 1 to |S_j| do
11:             if i is in a bin in S then
12:                 S_k ← S_k \ {S_j}
13:                 break
14: return S
```

Figure 5.4 shows the result of the AGX operator on our two example parent solutions. Here, $\mathcal{S}_1$ is the initial parent solution as it contains the fullest bin of both parent solutions (bin S_1). The resultant partial offspring $\mathcal{S}$ contains five bins, and four items remain unpacked.

It is evident that if the bins of a solution are well-packed then fewer bins will be required overall to pack all of the items. These bins are the ones that contribute to the fitness of the solution, and therefore AGX aims to preserve the most well-filled bins by prioritising them in the selection of bins from each parent solution.

From another perspective, our AGX operator can be considered similar to the GPX operator of Galinier and Hao (1999), which was proposed for the graph colouring problem; however in the latter only the duplicate elements themselves are removed from the groups. As we have seen, removing individual items from bins in

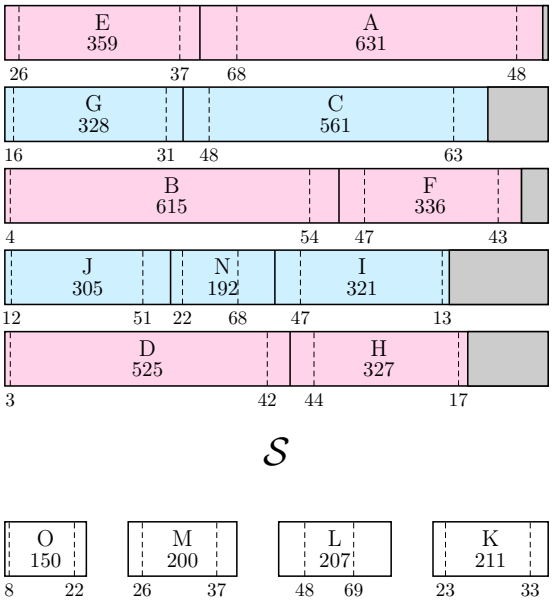


Figure 5.4: The partial offspring $\mathcal{S}$ created using the AGX recombination operator, where parent solution $\mathcal{S}_1$ is the starting solution as it contains the fullest bin (S_1). Bins S_1, S_2, and S_4 have been inserted from parent $\mathcal{S}_1$ and bins S_4 and S_5 have been inserted from parent $\mathcal{S}_2$, resulting in four unpacked items.

the SCPP may render the bins infeasible as the remaining subset of items may not be able to satisfy the vicinal sum constraint. Therefore, we opt to remove entire bins to ensure the bins remain feasible.

5.2.3 The Alternating Grouping Crossover Using Bin Cardinality

Our final operator, the alternating grouping crossover using bin cardinality (AGX′), performs in a similar manner to AGX; however rather than choosing the fullest bin to insert into the offspring solution, AGX′ selects bins containing the most items, $\max_{S_j \in \mathcal{S}_k}(|S_j|)$ (where $\mathcal{S}_k$ is the parent solution under consideration, $k \in \{1, 2\}$). This method has the ability to preserve bins containing items that may be seen as harder to pack alongside other items. The pseudocode for AGX′ is provided in Algorithm 9, where Lines 4 and 6 differ from the previous algorithm in that the highest cardinality bin is selected. If the cardinality of the best bin from each parent is equal, the fuller bin of the two is chosen, breaking ties randomly.

The resulting offspring solution produced using AGX′ on our example parent solutions is shown in Figure 5.5. Again, $\mathcal{S}_1$ is chosen as the starting parent solution as it contains the bin with the most items in both parent solutions (bin S_6). This operator creates a partial offspring comprising just four bins, unlike the partial offspring produced by GGA and AGX, and leaves four items unpacked.

Again, similar to the AGX operator, the premise of AGX′ lies in the idea that if more items are packed into each bin then fewer bins will be required in a solution to

Algorithm 9 AGX′ ($\mathcal{S}_1$, $\mathcal{S}_2$)

$\mathcal{S} \leftarrow \emptyset$
$k \leftarrow 0$
$q \leftarrow \min(|\mathcal{S}_1|, |\mathcal{S}_2|) - 1$
let $\mathcal{S}_k$ be the parent solution containing the bin with the most items
while $|\mathcal{S}| < q$ **or** $\mathcal{S}_k = \emptyset$ **do**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \max_{S \in \mathcal{S}_k}(|S|)$ ▷ Add the highest cardinality bin in $\mathcal{S}_k$ to $\mathcal{S}$
    if $k = 1$ **then** $k \leftarrow 2$
    else if $k = 2$ **then** $k \leftarrow 1$
    for $j \leftarrow 1$ **to** $|\mathcal{S}_k|$ **do**
        for $i \leftarrow 1$ **to** $|S_j|$ **do**
            if i is in a bin in $\mathcal{S}$ **then**
                $\mathcal{S}_k \leftarrow \mathcal{S}_k \backslash \{S_j\}$
                break
return $\mathcal{S}$

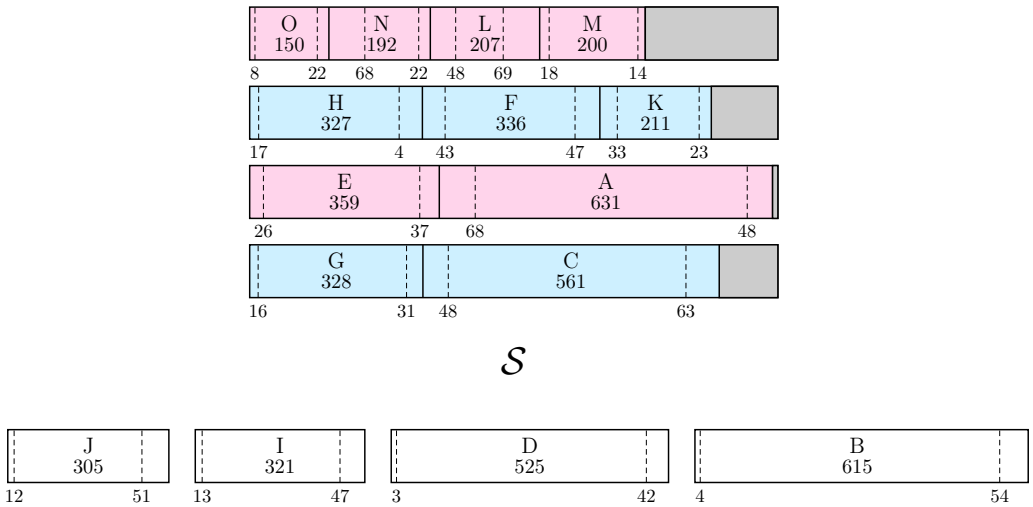


Figure 5.5: The partial offspring $\mathcal{S}$ created using the AGX′ recombination operator, where parent solution $\mathcal{S}_1$ is the starting solution as it contains the bin with the most items (S_6). Bins S_6 and S_1 have been inserted from parent $\mathcal{S}_1$ and bins S_1 and S_4 have been inserted from parent $\mathcal{S}_2$, resulting in four unpacked items.

pack all items. For the SCPP, it can be more difficult to pack a larger number of items into a single bin in comparison to the classical BPP, as this increases the number of adjacent score widths that have to fulfil the vicinal sum constraint. Therefore, the aim of AGX′ is to protect bins that contain feasible configurations of many items. Note that for a bin to contain more items, the items will usually be of a smaller width so as not to exceed the bin capacity. Consequently, the remaining unpacked items will tend to be of larger widths.

5.2.4 Repair Operator

As previously explained, removing individual items from bins may cause a violation of the vicinal sum constraint; thus to ensure offspring feasibility our recombination operators need to disregard entire bins that contain duplicate items in the parent solutions (i.e. items that have already been added to the offspring solution $\mathcal{S}$). However, these bins may also possess items that are not yet present in $\mathcal{S}$, and so on completion of the recombination the offspring solution may not contain all items in $\mathcal{I}$. To repair these partial solutions, the following operator is implemented. First, the MFFD$^+$ heuristic presented in Chapter 4 is applied using just the unpacked items to form a second partial solution $\mathcal{S}'$. The partial solutions $\mathcal{S}$ and $\mathcal{S}'$ are then used as input into a local search procedure, described in the following section, to create a full feasible offspring solution.

5.3 Local Search

Local search is a method used to improve a solution by iteratively making small changes to the solution in search of a better quality neighbouring solution; thus, local search methods have the ability to move between solutions in the search space. Eventually, a solution will hopefully be found that is of higher quality than all neighbouring solutions – a so-called local optimum. As the three recombination operators detailed in the previous section yield partial offspring solutions comprising entire bins inherited from parent solutions, it is worthwhile incorporating a local search procedure for a more fine-grained search on individual solutions.

Our local search procedure uses two partial solutions, $\mathcal{S}$ and $\mathcal{S}'$, containing bins that together make up a full solution comprising all items in $\mathcal{I}$. The procedure operates by strategically shuffling items between the bins of $\mathcal{S}$ and $\mathcal{S}'$. This method is based on the dominance criterion of Martello and Toth (1990b) used for the BPP: if a bin S_x *dominates* a bin S_y, then a solution containing S_x will have no more bins than a solution containing S_y. In particular, the aim is to move larger items into $\mathcal{S}$ such that the fullness $A(S)$ of the bins $S \in \mathcal{S}$ increases, whilst the number of items per bin is maintained or decreases, improving the quality of these bins. Simultaneously, this causes smaller items to be moved into bins in $\mathcal{S}'$, which will potentially be easier to repack into bins in $\mathcal{S}$ at a later stage.

The local search procedure, shown in Algorithm 10, takes two feasible partial solutions, $\mathcal{S}$ and $\mathcal{S}'$, permutes the bins of both, and then attempts to move items between the two partial solutions in four stages:

(i) swapping a pair of items from a bin in $\mathcal{S}$ with a pair of items from a bin in $\mathcal{S}'$ (Lines 4–13);

(ii) swapping a pair of items from a bin in $\mathcal{S}$ with an individual item from a bin in $\mathcal{S}'$ (Lines 14–23);

(iii) swapping individual items from bins in $\mathcal{S}$ and $\mathcal{S}'$ (Lines 24–33); and

(iv) moving an item from a bin in $\mathcal{S}'$ to a bin in $\mathcal{S}$ (Lines 34–43).

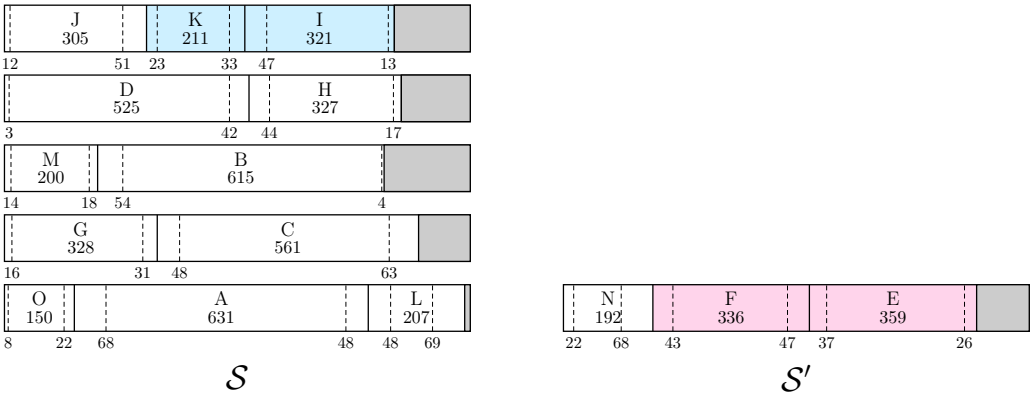
Algorithm 10 LocalSearch ($\mathcal{S}$, $\mathcal{S}'$)

```
1: permute the bins of each partial solution S and S'
2: repeat
3:     changed ← false
4:     for x ← 1 to |S| do                                                    ▷ Stage (i)
5:         for each pair of items i, j in bin S_x in S do
6:             for y ← 1 to |S'| do
7:                 for each pair of items k, l in bin S_y in S' do
8:                     if w_i + w_j < w_k + w_l and A(S_x) − (w_i + w_j) + (w_k + w_l) ≤ W then
9:                         S'_x ← AHC(S_x, k, l), S'_y ← AHC(S_y, i, j)
10:                        if S'_x ∈ F and S'_y ∈ F then
11:                            S_x ← S'_x, S_y ← S'_y
12:                            changed ← true
13:                            go to Line 14
14:    for x ← 1 to |S| do                                                    ▷ Stage (ii)
15:        for each pair of items i, j in bin S_x in S do
16:            for y ← 1 to |S'| do
17:                for each item k in bin S_y in S' do
18:                    if w_i + w_j < w_k and A(S_x) − (w_i + w_j) + w_k ≤ W then
19:                        S'_x ← AHC(S_x, k), S'_y ← AHC(S_y, i, j)
20:                        if S'_x ∈ F and S'_y ∈ F then
21:                            S_x ← S'_x, S_y ← S'_y
22:                            changed ← true
23:                            go to Line 24
24:    for x ← 1 to |S| do                                                    ▷ Stage (iii)
25:        for each item i in bin S_x in S do
26:            for y ← 1 to |S'| do
27:                for each item k in bin S_y in S' do
28:                    if w_i < w_k and A(S_x) − w_i + w_k ≤ W then
29:                        S'_x ← AHC(S_x, k), S'_y ← AHC(S_y, i)
30:                        if S'_x ∈ F and S'_y ∈ F then
31:                            S_x ← S'_x, S_y ← S'_y
32:                            changed ← true
33:                            go to Line 34
34:    for y ← 1 to |S'| do                                                   ▷ Stage (iv)
35:        for each item k in bin S_y in S' do
36:            for x ← 1 to |S| do
37:                if A(S_x) + w_k ≤ W then
38:                    S'_x ← AHC(S_x, k), S'_y ← AHC(S_y\{k})
39:                    if S'_x ∈ F and S'_y ∈ F then
40:                        S_x ← S'_x, S_y ← S'_y
41:                        if S_y = ∅ then S' ← S'\{S_y}
42:                        changed ← true
43:                        go to Line 44
44: until not changed or S' = ∅
45: if S' ≠ ∅ then
46:     S'' ← MFFD+(S')
47:     if |S''| ≤ |S'| then
48:         S ← S ∪ S''
49:     else S ← S ∪ S'
50: return S
```

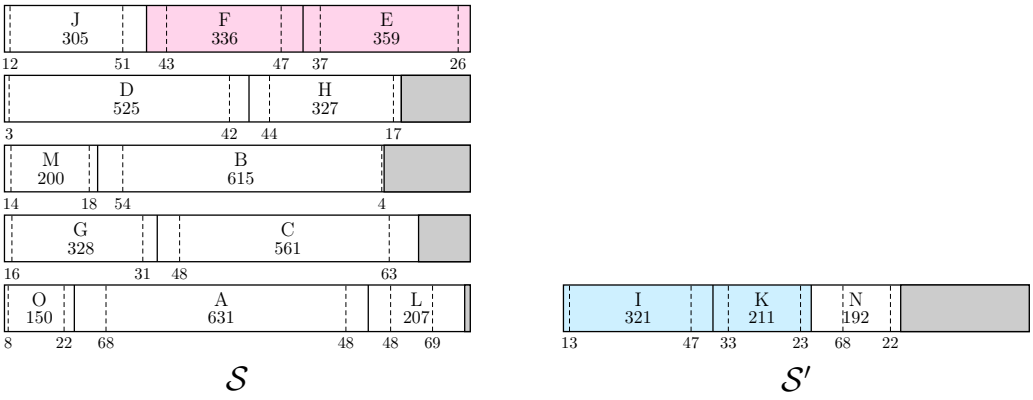
During Stages (i)–(iii), the total width of the item(s) selected from $\mathcal{S}'$ must exceed the total width of the item(s) selected from $\mathcal{S}$. AHC is then used to determine whether the new bins are members of $\mathcal{F}$. If AHC returns a feasible configuration for both of the modified bins then the swap or move is performed; else the next set of

items are assessed. As shown, once an exchange occurs the procedure immediately proceeds to the next stage. This process is repeated until all four stages have been executed in succession with no changes to $\mathcal{S}$ or $\mathcal{S}'$, or until no items remain in $\mathcal{S}'$ (Line 44). At this point, any items that remain in $\mathcal{S}'$ are copied and MFFD$^+$ is executed to generate a new feasible partial solution $\mathcal{S}''$ (Line 46). Then, $\mathcal{S}'$ and $\mathcal{S}''$ are compared and the partial solution comprising the fewest bins is copied into $\mathcal{S}$ to form a full, feasible solution (Lines 47–49). Therefore, this procedure cannot increase the number of bins in a solution as no new bins are opened; however it does have the potential to decrease the number of bins.

Figure 5.6 shows an example of Stage (i) of our local search procedure as part of the repair operator on the GGA output from Figure 5.3. Here, MFFD$^+$ has beeen applied to the unpacked items to produce the partial solution $\mathcal{S}'$, and the bins of $\mathcal{S}$ have been permuted. Local search has then found a pair of items in $\mathcal{S}$, K and I, and a pair of items in $\mathcal{S}'$, F and E, where the total width of the items in $\mathcal{S}'$ exceeds the total width of the items in $\mathcal{S}$. Since AHC is able to find a feasible configuration for



(a) Partial solutions $\mathcal{S}$ and $\mathcal{S}'$ produced using GGA and MFFD$^+$, where the pairs of items to be exchanged are highlighted.



(b) Partial solutions $\mathcal{S}$ and $\mathcal{S}'$ after the pairs of items have been exchanged.

Figure 5.6: Stage (i) of the local search procedure applied to partial solutions from the GGA output in Figure 5.3, where MFFD$^+$ has been used on the missing items to produce $\mathcal{S}'$. It can be seen that after the exchange the bin in $\mathcal{S}$ is fuller.

the new subsets of items in each bin, the items can be exchanged successfully. As a result the bin in $\mathcal{S}$ is now fuller, using the entire capacity of the bin, whilst $\mathcal{S}'$ now contains smaller items which the local search procedure will attempt to move into $\mathcal{S}$ in subsequent iterations.

The total running time of our local search procesure is $O(n^6)$, including instances of AHC. However, this is a very conservative bound, as in many cases AHC will not need to be executed. Furthermore, the value of n here in fact refers to the number of items in a single bin, as opposed to the total number of items in $\mathcal{I}$, which is of course significantly higher.

Variations of this method for the BPP can be seen in Lewis (2009), Lewis and Holborn (2017), Falkenauer (1996), and Levine and Ducatelle (2004); however the addition of the vicinal sum constraint results in fewer changes than seen in these previous implementations. By iterating through the different stages, numerous distinct subsets of items in the bins are produced, generating more possibilities for feasible orderings of items.

5.4 Mutation

The purpose of mutation is to allow the introduction of new bin packings to the population, maintaining genetic diversity and assuring that the entire search space is connected, thus preventing the premature convergence of the population (Buckles and Petry, 1992). The mutation of a solution $\mathcal{S}$ in our EA consists of permuting the bins and inserting $1 < r < |\mathcal{S}|$ bins selected randomly from $\mathcal{S}$ into a set $\mathcal{S}'$. Local search is then executed using these two partial solutions to produce a full feasible solution $\mathcal{S}$. In our case, this mutation operator also has the ability to improve the fitness value of a solution and even potentially reduce the number of bins in a solution.

Similar mutation operators proposed for the BPP make use of values of r between three and five (Falkenauer, 1998; Levine and Ducatelle, 2004; Singh and Gupta, 2007) and it has also been suggested that r should be a random variable calculated relative to the number of bins $|\mathcal{S}|$ and the number of bins that are not completely full (Quiroz-Castellanos et al., 2015). These authors also favour selecting emptier bins, however it may be beneficial to separate items within fuller bins, and there is also the potential for an emptier bin to obtain a high quality packing with the addition of other items. Lewis and Holborn (2017) therefore opted for a value for r according to the distribution $R \sim 1 + \mathrm{B}(|\mathcal{S}|, 3/|\mathcal{S}|)$. Due to the novelty of the SCPP, for our mutation operator we decided on simply selecting r uniform randomly to prevent bias whilst also ensuring that each partial solution comprises at least one bin, which

we have found generated high quality solutions.

5.5 Solution Fitness

In EAs, fitness functions are used to measure the quality of each individual candidate solution. For the classical one-dimensional BPP, a tailored function can be used to determine the fitness of a solution as opposed to simply relying on the number of bins within a solution. The reason for this is two-fold: firstly, given two solutions with an equal number of bins, it is impossible to determine the fitter solution based on the number of bins alone. Secondly, we note that the fitness of a solution not only depends on the number of bins used, but also *how* the items are packed into the bins. It is clear that if the bins' capacities are utilised more fully, fewer bins will be needed to pack all items. A solution comprising fuller bins may also contain bins that are nearly empty, which is beneficial as it allows further items to be packed, or for the residual material to be used for other means.

Falkenauer and Delchambre (1992) make use of the following function to calculate the fitness of a solution $\mathcal{S}$ of the BPP:

$$f(\mathcal{S}) = \frac{\sum_{S_j \in \mathcal{S}} (A(S_j)/W)^2}{|\mathcal{S}|} \,, \tag{5.1}$$

which assigns higher values to fitter solutions. Note also that under certain circumstances, it can be guaranteed with the BPP that if $|\mathcal{S}_1| < |\mathcal{S}_2|$ then $f(\mathcal{S}_1) > f(\mathcal{S}_2)$. Hence, a global optimum for this fitness function is associated with the optimal solution containing the fewest number of bins. Specifically, this condition holds when it can be guaranteed that the solution has at most one bin that is less than half-full. This is easy to ensure in the BPP, for example, by using the FF heuristic or a similar variant, as the contents of two less than half-full bins can be combined into a single bin. This is not the case for the SCPP, however, as the items of two half-full bins may not be able to form a single feasible packing due to the vicinal sum constraint. Hence, solutions for the SCPP may well contain multiple bins that are less than half-full. As a result, we have observed a number of cases of the SCPP where although $|\mathcal{S}_1| < |\mathcal{S}_2|$, the corresponding fitness values do not conform to $f(\mathcal{S}_1) > f(\mathcal{S}_2)$. Consequently, we cannot rely on the fitness function alone to guide us towards an optimal solution, as this may lead to a final solution comprising more bins than necessary. We therefore use the number of bins to indicate the quality of a solution, and only use the fitness function (5.1) when comparing solutions containing an equal number of bins. For example, the parent solutions $\mathcal{S}_1$ and $\mathcal{S}_2$ in Figure 5.1 comprise the same number of bins; however $f(\mathcal{S}_1) = 0.777$ whilst $f(\mathcal{S}_2) = 0.774$, and so $\mathcal{S}_1$ is considered the fitter solution. Although there exist a variety of fitness functions in literature,

we opted for this particular function due to its simplicity and its consideration into the quality of packings in individual bins.

5.6 The Evolutionary Algorithm Framework

We now present our evolutionary algorithm framework for the SCPP, shown in Algorithm 11. The process begins by generating a population $\mathcal{P}$ comprising $|\mathcal{P}|$ candidate solutions for the given instance $\mathcal{I}$ of the SCPP, where one solution is created using MFFD$^+$ and the remaining $|\mathcal{P}| - 1$ solutions are created using MFFR$^+$ (where the items are packed in a random order). Each solution is mutated before being added to the population. The best-so-far solution, $\mathcal{S}_{\text{bsf}}$, is initially set to the best solution in $\mathcal{P}$ (Line 2).

Each iteration of our EA involves selecting two parent solutions $\mathcal{S}_1$ and $\mathcal{S}_2$ from the population at random and applying one of our three recombination operators to build an offspring solution $\mathcal{S}$ (Lines 5–6). If $\mathcal{S}$ does not contain all items in $\mathcal{I}$, MFFD$^+$ is applied to the unpacked items to create a second partial solution $\mathcal{S}'$, and our local search procedure is executed on the partial solutions $\mathcal{S}$ and $\mathcal{S}'$ to produce a full offspring solution $\mathcal{S}$ (Lines 7–9). The full feasible offspring solution $\mathcal{S}$ is then mutated and inserted into the population $\mathcal{P}$, replacing the least fit of its parents (Lines 10–12). Finally, if the offspring $\mathcal{S}$ is fitter than the current best-so-far solution $\mathcal{S}_{\text{bsf}}$, $\mathcal{S}$ is taken as the new best-so-far solution (Lines 13–14). The process is repeated until a specified time limit is reached (Line 3).

Algorithm 11 EA($\mathcal{I}$)

```
1:  create initial population P of candidate solutions
2:  S_bsf ← best solution in P
3:  while time limit not reached do
4:      S ← ∅
5:      randomly select two parent solutions S_1 and S_2 from P
6:      S ← Recombination(S_1, S_2)          ▷ Using either GGA, AGX, or AGX′
7:      if ⋃_{S∈S} ≠ I then
8:          S′ ← MFFD+(I\S)
9:          S ← LocalSearch(S, S′)
10:     S ← Mutate(S)
11:     P ← P\{least fit parent solution}
12:     P ← P ∪ {S}
13:     if S is better than S_bsf then
14:         S_bsf ← S
15: return S_bsf
```

Although there is the option of using the Pair-Smallest (PS) heuristic for the SCPP presented in the previous chapter, the design of PS means that only one solution

can be produced for a given instance $\mathcal{I}$ of the SCPP. In contrast, by altering the order of the items we can apply the MFFD$^+$ heuristic (i.e. MFFR$^+$ as the items are in random order) to obtain numerous distinct solutions for the same instance $\mathcal{I}$, allowing us to produce a diverse population $\mathcal{P}$ of solutions.

5.7 Computational Results

To ensure a fair comparison with the heuristics in Chapter 4, the same set of problem instances generated for the previous experiments were used to test our EA. After preliminary experiments, we settled on an initial population containing $|\mathcal{P}| = 25$ candidate solutions, which provides a suitable balance between population diversity and convergence. Across all instances, the EA was given a fixed time limit of 600 seconds, which was deemed to be sufficient for these trials. Tables 5.1 to 5.3 display the results obtained from the EA experiments using the different recombination operators and bin sizes. A full breakdown of these results can be found online along with all of our source code (Hawa, 2020e).[2]

For all values of δ, as $|\mathcal{I}|$ and W increase the local search procedure and mutation operator take longer as the rise in the number of bins in solutions and/or the number of items per bin results in an increase in the number of applications of AHC; thus fewer iterations of the EA are performed. For example, the average number of iterations range from 85,107 to 3,171,066 when $W = 1250$ for artificial instance using 1000 and 100 items respectively when $\delta = 0.5$, whilst the corresponding figures for real instances using $W = 5000$ are 105 and 97,528 iterations. Consequently, we see that the difference between $|\mathcal{S}|$ and the theoretical minimum t also increases as the number of items increases, more so for the larger bin sizes. The reduction in the number of EA iterations means that the evolution of the population is restricted by the running time, which prevents better solutions from being achieved. This suggests that the algorithm is less accurate.

The same trend seen in the previous experiments in Chapter 4 is evident throughout the results where the average number of bins $|\mathcal{S}|$ and the difference between $|\mathcal{S}|$ and t decreases as δ increases due to higher proportions of score widths fulfilling the vicinal sum constraint. Furthermore, the coefficient of variation is also higher for real instances as with the previous results because of the weakly heterogeneous item sets.

Recall that the theoretical minimum t (2.2) for the BPP used to compare the results is often not accurate for the SCPP, as explained in Chapter 2 (see Section 2.8). Therefore, when $\delta = 0.25$, of the 18 instance classes we see that only one class has

[2]As with the experiments conducted in Chapter 4, all experiments were implemented in C++ and executed on Windows machines with Intel Core i5-6500 3.20GHz processors and 8GB of RAM.

Table 5.1: Best solutions obtained from the EA using the GGA, AGX, and AGX′ recombination operators for $\delta = 0.25$. Figures in bold indicate the best results for each instance class. Asterisks indicate statistical significance at $\leq 0.05(^{*})$ and $\leq 0.01(^{**})$ according to a two-tailed paired t-test and two-tailed McNemar's test for the $|\mathcal{S}|$ and $\%t$ columns respectively.

$\delta = 0.25$			GGA		AGX		AGX′	
Type, W	$\lvert\mathcal{I}\rvert$	t^a	$\lvert\mathcal{S}\rvert^b$	$\%t^c$	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$
a, 1250	100	46.13	****53.34** ± 6.7	0.0	53.50 ± 6.6	0.0	53.38 ± 6.6	0.0
	500	229.37	****256.78** ± 2.4	0.0	260.85 ± 2.6	0.0	257.92 ± 2.5	0.0
	1000	458.37	****515.24** ± 1.5	0.0	520.96 ± 1.7	0.0	515.46 ± 1.7	0.0
a, 2500	100	23.32	36.21 ± 11.1	0.0	35.57 ± 11.7	0.0	****35.23** ± 12.0	0.0
	500	114.94	183.83 ± 4.7	0.0	183.53 ± 4.7	0.0	****181.68** ± 4.9	0.0
	1000	229.44	372.44 ± 3.5	0.0	375.22 ± 3.5	0.0	****369.52** ± 3.5	0.0
a, 5000	100	11.92	35.20 ± 12.7	0.0	35.14 ± 12.7	0.0	****35.09** ± 12.7	0.0
	500	57.72	****173.42** ± 5.7	0.0	174.31 ± 5.6	0.0	173.67 ± 5.6	0.0
	1000	114.97	349.45 ± 4.3	0.0	350.91 ± 4.3	0.0	**349.41** ± 4.3	0.0
r, 1250	100	46.44	61.72 ± 16.7	0.0	61.79 ± 16.6	0.0	****61.70** ± 16.7	0.0
	500	229.97	304.18 ± 15.9	0.0	305.04 ± 15.7	0.0	****303.65** ± 15.9	0.0
	1000	459.38	609.12 ± 15.8	0.0	609.97 ± 15.7	0.0	****608.20** ± 15.8	0.0
r, 2500	100	23.47	42.71 ± 26.1	**1.1**	42.82 ± 25.9	1.0	****42.64** ± 26.2	**1.1**
	500	115.24	***214.88** ± 23.6	0.0	217.01 ± 23.2	0.0	215.09 ± 23.5	0.0
	1000	229.95	**435.13** ± 23.3	0.0	437.82 ± 23.0	0.0	435.33 ± 23.1	0.0
r, 5000	100	11.98	42.41 ± 29.5	0.0	41.40 ± 29.5	0.0	**41.39** ± 29.5	0.0
	500	57.87	****207.38** ± 27.0	0.0	208.13 ± 26.7	0.0	207.69 ± 26.8	0.0
	1000	115.23	**420.36** ± 26.3	0.0	421.26 ± 26.1	0.0	420.41 ± 26.2	0.0

[a] $t = \lceil \sum_{i=1}^{n} w_i / W \rceil$ (mean from 1000 instances).
[b] Number of bins per solution (mean from 1000 instances plus or minus the coefficient of variation (%)).
[c] Percentage of instances in which the solution comprises t bins.

instances for which our EA is able to find solutions comprising t bins, whilst when $\delta = 0.75$ our EA is able to find solutions with t bins for instances in 16 classes. For two instance classes, solutions comprising t bins have been found for all 1000 problem instances. However, as we are not able to determine the lower bounds for the fewest number of bins for these instances, it is possible that in many of these cases our EA has in fact been able to find optimal solutions.

Focusing on Table 5.1, we see that the GGA and AGX′ recombination operators perform well, with GGA producing solutions with the fewest number of bins on average for eight instance classes, and AGX′ for 10 instance classes. The design of AGX′ prioritising bins containing the *most* items seems to be beneficial for the smallest value of δ as the vicinal sum constraint is harder to fulfil; thus finding collections of individual bins containing many items will clearly be advantageous. In five of the instance classes where AGX′ outperforms GGA, the average number of bins produced by AGX′ ranges from 0.92 to 2.92 fewer bins than the corresponding GGA solutions, whilst there is only one instance in which GGA produces solutions on average using over 0.31 fewer bins than AGX′. The AGX recombination operator is not favourable in this scenario, as it operates based on bin fullness. This could be rectified by choosing bins that contain a small number of large items, which would

Table 5.2: Best solutions obtained from the EA using the GGA, AGX, and AGX′ recombination operators for $\delta = 0.5$. Figures in bold and asterisks should be interpreted as in Table 5.1.

$\delta = 0.5$			GGA		AGX		AGX′	
Type, W	$\lvert\mathcal{I}\rvert$	t	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$
a, 1250	100	46.13	**47.51** ± 6.6	**27.4**	47.53 ± 6.6	25.3	**47.51** ± 6.6	26.8
	500	229.37	****230.63** ± 2.3	****19.6**	230.89 ± 2.3	7.7	231.36 ± 2.2	0.2
	1000	458.37	****459.80** ± 1.5	**3.3**	459.93 ± 1.5	3.0	461.84 ± 1.4	0.0
a, 2500	100	23.32	**23.33** ± 4.5	99.4	**23.33** ± 4.5	**99.9**	**23.33** ± 4.5	99.4
	500	114.94	****115.13** ± 1.9	****80.9**	115.43 ± 1.9	59.7	115.31 ± 1.9	63.6
	1000	229.44	****230.18** ± 1.4	****34.0**	231.36 ± 1.4	9.8	230.66 ± 1.4	11.4
a, 5000	100	11.92	****12.16** ± 9.2	****90.5**	12.20 ± 9.4	88.4	12.19 ± 9.1	82.0
	500	57.72	****58.87** ± 3.0	****38.8**	59.92 ± 3.7	16.8	59.69 ± 3.6	21.2
	1000	114.97	****119.02** ± 2.7	****3.7**	121.37 ± 3.2	0.2	120.78 ± 3.0	0.5
r, 1250	100	46.44	**53.47** ± 18.4	**1.7**	53.52 ± 18.3	1.6	53.48 ± 18.3	1.4
	500	229.97	**262.96** ± 17.5	0.0	263.81 ± 17.2	0.0	262.97 ± 17.4	0.0
	1000	459.38	**526.00** ± 17.4	0.0	527.44 ± 17.1	0.0	526.21 ± 17.2	0.0
r, 2500	100	23.47	25.51 ± 20.9	**66.9**	25.57 ± 21.1	65.2	****25.45** ± 20.9	66.5
	500	115.24	****127.92** ± 20.5	18.2	129.85 ± 20.9	**18.5**	128.78 ± 20.3	7.4
	1000	229.95	****259.62** ± 21.0	1.6	262.54 ± 21.2	****3.0**	261.36 ± 20.8	0.0
r, 5000	100	11.98	****17.21** ± 48.2	****53.4**	17.25 ± 48.0	52.0	17.26 ± 48.0	50.6
	500	57.87	****86.91** ± 44.6	****15.1**	88.73 ± 43.9	11.5	88.25 ± 43.9	11.6
	1000	115.23	****180.08** ± 44.1	****5.9**	182.57 ± 43.7	4.3	181.72 ± 43.6	3.5

Table 5.3: Best solutions obtained from the EA using the GGA, AGX, and AGX′ recombination operators for $\delta = 0.75$. Figures in bold and asterisks should be interpreted as in Table 5.1.

$\delta = 0.75$			GGA		AGX		AGX′	
Type, W	$\lvert\mathcal{I}\rvert$	t	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$
a, 1250	100	46.13	**47.39** ± 6.6	**31.8**	47.40 ± 6.6	31.2	47.40 ± 6.6	31.4
	500	229.37	****230.52** ± 2.3	****25.3**	230.62 ± 2.3	18.6	230.83 ± 2.2	4.6
	1000	458.37	****459.45** ± 1.5	****18.5**	459.53 ± 1.5	14.4	460.64 ± 1.4	0.0
a, 2500	100	23.32	23.33 ± 4.5	99.9	**23.32** ± 4.5	**100.0**	23.33 ± 4.5	99.6
	500	114.94	****114.96** ± 1.9	****98.5**	114.97 ± 1.9	96.9	115.21 ± 1.9	73.0
	1000	229.44	229.54 ± 1.4	89.7	****229.51** ± 1.4	****93.0**	230.21 ± 1.4	22.5
a, 5000	100	11.92	**11.92** ± 4.8	**100.0**	**11.92** ± 4.8	**100.0**	**11.92** ± 4.4	**100.0**
	500	57.72	****57.72** ± 2.0	****99.8**	57.73 ± 2.0	98.9	57.75 ± 1.9	96.9
	1000	114.97	****115.02** ± 1.4	****94.7**	115.05 ± 1.4	91.9	115.14 ± 1.4	82.7
r, 1250	100	46.44	**52.23** ± 18.5	**3.2**	52.24 ± 18.5	3.1	**52.23** ± 18.5	3.1
	500	229.97	****256.91** ± 17.5	0.0	257.21 ± 17.4	0.0	257.06 ± 17.5	0.0
	1000	459.38	****513.24** ± 17.5	0.0	513.95 ± 17.4	0.0	513.80 ± 17.3	0.0
r, 2500	100	23.47	**23.56** ± 11.7	**93.9**	**23.56** ± 11.7	93.7	23.58 ± 11.7	92.1
	500	115.24	****116.07** ± 11.1	58.5	116.22 ± 11.3	**60.3**	116.55 ± 11.2	24.9
	1000	229.95	****232.37** ± 11.2	19.3	232.83 ± 11.3	****29.0**	233.60 ± 11.3	0.8
r, 5000	100	11.98	**12.11** ± 14.4	**97.5**	12.12 ± 14.4	97.1	**12.11** ± 14.4	97.1
	500	57.87	****59.10** ± 15.5	****80.5**	59.39 ± 16.2	74.7	59.35 ± 16.0	71.7
	1000	115.23	****118.71** ± 16.9	****49.7**	119.11 ± 17.4	42.5	119.17 ± 17.3	36.3

then mean that many more bins overall will be needed to pack the remaining items.

Moving on to the higher values of δ in Tables 5.2 and 5.3, we see that the GGA recombination operator dominates the results, outperforming the AGX and AGX′ recombination operators in the majority of instance classes. For GGA, the offspring $\mathcal{S}$ will always contain a group of bins from parent solution $\mathcal{S}_2$, and potentially additional bins from parent solution $\mathcal{S}_1$. In contrast, AGX and AGX′ select individual bins

from each parent solution in turn, prioritising what are perceived to be high quality bins. The items contained in a single bin selected from one parent solution may be dispersed amongst numerous bins in the other parent solution; thus all offending bins will need to be removed. Some of these eliminated bins may be well-packed bins, resulting in lower quality or less desirable bins remaining in the other parent solution. Although this is also possible with GGA, it appears that the lack of bias when selecting groups of bins makes the operator more suitable, particularly for these higher values of δ.

Furthermore, recall that the offspring $\mathcal{S}$ produced by the AGX and AGX$'$ recombination operators will comprise at most $\min(|\mathcal{S}_1|, |\mathcal{S}_2|) - 1$ bins; if a large number of bins are removed from the parent solutions the offspring may potentially contain very few bins on completion of the recombination. Consequently, many items in $\mathcal{I}$ will not be present in the offspring and the repair operator will need to be executed. Thus, in this case, AGX and AGX$'$ may resemble large mutation operators as opposed to recombination operators. In addition, for these higher values of δ where a higher proportion of score widths meet the vicinal sum constraint, selecting groups of bins at random as with GGA rather than choosing individual bins further promotes diversity, encouraging wider areas of the solution space to be explored.

5.8 Summary

In this chapter we presented a novel evolutionary algorithm (EA) tailored for the SCPP. Firstly, we introduced three distinct recombination operators: GGA, an adaptation of a method proposed by Falkenauer and Delchambre (1992); AGX, which focuses on selecting fuller bins; and AGX$'$, which prioritises bins containing a large number of items. We then went on to describe our local search procedure, mutation operator, and fitness function, before outlining how these stages are utilised within our EA framework as a whole.

As previously discussed, a prominent reason for designing an evolutionary algorithm for the SCPP is due to the often inordinately large number of feasible packings, which can become too large to enumerate. In preliminary tests we also attempted to maintain two sets, $\mathcal{X}$ and $\mathcal{Y}$, throughout our EA program where $\mathcal{X}$ contained subsets of items that our AHC algorithm determined could be feasibly packed into a single bin, and $\mathcal{Y}$ contained subsets of items for which AHC could not find a feasible arrangement. These sets were implemented as binary trees, and so packings could be searched for and added to the sets in logarithmic time. Thus, for any subset of items AHC would only have to be executed at most once. This would have been particularly useful for real instances where many items have the same dimensions and

identical packings arise more frequently. However, during these preliminary tests we saw that the sizes of $\mathcal{X}$ and $\mathcal{Y}$ often grew unmanageably large such that our machines were unable to maintain and store the sets.

Nevertheless, we saw that using an EA produced superior solutions on average in the majority of instances classes in comparison to the previous heuristics. Utilising an EA allows us to incorporate a variety of operators specifically designed for the SCPP, including the AHC algorithm for instances of the sub-SCPP. Furthermore, the local search procedure introduces the ability to move items *between* bins, a useful property which our heuristics do not possess.

In our EA, the only exact method used is the AHC algorithm for instances of the sub-SCPP. Therefore, in the next chapter, we explore how an exact solver can be used for the SCPP along with operators from our EA.

Chapter 6

Combining Metaheuristics and Exact Methods for the Score-Constrained Packing Problem

In this chapter, we investigate the effects of combining elements of our evolutionary algorithm (EA) from the previous chapter with an exact method for the SCPP in various formats. To begin, we present an integer linear programming (ILP) formulation for the SCPP and provide a brief literature review on the topic of combining different optimisation techniques. We then define and formulate the Minimum Cardinality Exact Cover Problem (MXCP) and discuss its relation to the SCPP. Next, the Generate and Solve (GS) framework is introduced, along with a specific instantiation of GS known as the Construct, Merge, Solve & Adapt (CMSA) algorithm. Then, we present and explain two bespoke CMSA-based algorithms adapted for the SCPP, which combine our exact cover formulation with features from our EA. To finish, the results from extensive experiments conducted on our CMSA algorithms are analysed and compared.

6.1 Introduction

Let us begin by expressing the SCPP as an ILP from the perspective of a generalised vehicle routing problem (VRP). The problem is as follows: a fleet of vehicles are to leave from a depot, indexed by 0, to deliver to set of n customers and return to the depot. The distance between two customers i and j is denoted w_{ij} for all $i, j = 1, \ldots, n, i \neq j$, whilst the distance between the depot and all customers is 0.

The total distance of the route travelled by each vehicle is limited to a fixed distance W. Furthermore, the n customers are presented in pairs – that is, each customer i is paired with another customer, $p(i)$, for $i = 1, \ldots, n$. The vehicles must deliver to these paired customers in succession. The task is to determine the fewest number of routes, and therefore vehicles, required to deliver to all n customers exactly once.

$$\text{minimise} \quad \sum_{i=1}^{n} x_{i0} \tag{6.1a}$$

$$\text{subject to} \quad \sum_{i=0}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n \tag{6.1b}$$

$$\sum_{j=0}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{6.1c}$$

$$x_{i,p(i)} + x_{p(i),i} = 1 \qquad i = 1, \ldots, n \tag{6.1d}$$

$$0 \leq q_i \leq W + (w_{ij} - W)x_{0i} \qquad i, j = 1, \ldots, n \tag{6.1e}$$

$$q_j \geq q_j + w_{ij} - W(1 - x_{ij}) + (W - 2w_{ij})x_{ji} \qquad i, j = 1, \ldots, n \tag{6.1f}$$

$$x_{ij} \leq \alpha_{ij} \qquad i, j = 0, 1, \ldots, n \tag{6.1g}$$

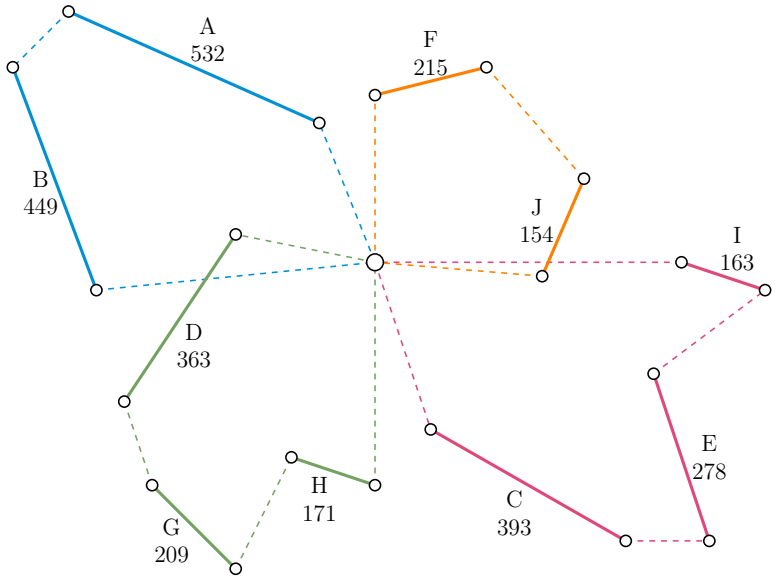
$$x_{ij} \in \{0, 1\} \qquad i, j = 0, 1, \ldots, n \tag{6.1h}$$

$$x_{ij} = \begin{cases} 1 & \text{if route goes from customer } i \text{ to customer } j \\ 0 & \text{otherwise.} \end{cases}$$

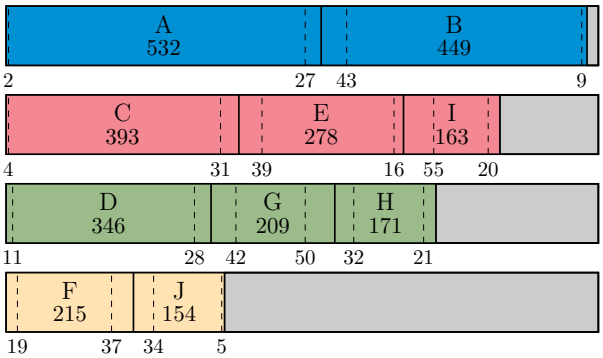
Here, the variable q_i is the total distance of a route up to customer i. The objective function (6.1a) is to minimise the number of routes by calculating the number of times a vehicle returns to the depot. Constraints (6.1b) and (6.1c) ensure that each customer is visited exactly once. Then, Constraint (6.1d) enforces the rule that each customer must be preceeded or succeeded by its paired customer in any given route. Constraints (6.1e) and (6.1f) impose the distance limit of the routes, whilst also preventing subroutes, i.e. routes that do not begin and end at the depot. Finally, Constraint (6.1g) states that all routes between customers must be valid routes, and Constraint (6.1h) restricts the variable x_{ij} to integers 0 or 1.

It is apparent that this problem is equivalent to the SCPP, where each pair of customers represents score widths from the same item and the maximum route distance is the bin capacity W. Then, $\alpha_{ij} = 1$ between customers i and j if the score widths are partners (i.e. on the same item) or are on different items and fulfil the

vicinal sum constraint. Thus, this generalised version of the vehicle routing problem is equivalent to the task of minimising the number of bins in a solution for a given instance of the SCPP such that the vicinal sum constraint is satisfied in each bin and no bin is overfilled. Figure 6.1 illustrates a solution to an example instance of this generalised VRP which translates to a solution for the corresponding SCPP, where the central vertex represents the depot whilst all other vertices represent customers. Here, the maximum route distance (and thus bin capacity) $W = 1000$, and the distances on the thicker lines of each route denote the items' widths. The distance between customers that are not paired, as well as distances between the depot and all other customers, are set to zero. Therefore, it can be seen that for the equivalent instance $\mathcal{I}$ of the SCPP a minimum of four bins are required to pack the items feasibly.



(a) A solution for an example instance of our generalised VRP consisting of four routes.



(b) A solution for an example instance of the SCPP comprising four bins.

Figure 6.1: An example showing a solution for an instance of our generalised VRP that translates to a solution for the corresponding SCPP instance. Here, the number of customers $n = 20$ and therefore $|\mathcal{I}| = 10$, the maximum route distance/bin capacity $W = 1000$, and $\tau = 70$.

Recall that the SCPP is NP-hard; finding optimal solutions will often take an unreasonable amount of time, especially for larger problem instances. In some preliminary experiments, we attempted to use the above ILP model to acquire optimal solutions for smaller SCPP instances.[1] Although solutions to problem instances of size $|\mathcal{I}| = 10$ were obtained in under a second, increasing $|\mathcal{I}|$ to just 15 resulted in running times of over four days. As noted, the problem instances used in this thesis for our experiments have a minimum size of 100 items; consequently, implementing an exact solver in this form for the SCPP is not a viable option. However, it may be useful to integrate exact methods into wider optimisation frameworks.

In Chapter 2 we provided an overview of metaheuristics and their applications to optimisation problems. The success of metaheuristics gave rise to the idea of combining metaheuristics with other optimisation techniques, often referred to as "hybrid metaheuristics", in the early 2000s with the first book dedicated to such methods published in 2008 (Blum et al.). The aim of merging various optimisation approaches into a single algorithm is to exploit the superior features of each method. Common implementations include integrating metaheuristics with constraint programming, dynamic programming, and branch and bound (Blum et al., 2011). Some combinations in particular have earned individual recognition such as memetic algorithms, which incorporate EAs and local search methods (Moscato et al., 1989) and matheuristics, which involve merging metaheuristics with mathematical programming techniques (Maniezzo et al., 2010).

The idea of consolidating multiple techniques has been widely studied for a range of grouping problems. For example, tabu search has been used with both a neighbourhood-based genetic algorithm (Nouri et al., 2016) and a variable neighbourhood descent approach (Chen and Chen, 2009) for machine scheduling problems. Many combinations of algorithms have also been proposed for variations of the VRP, including EAs and iterated local search (Bianchi et al., 2006), guided local search and tabu search (Tarantilis et al., 2009), simulated annealing and tabu search (Lin et al., 2009), and ant colony optimisation (ACO) and neighbourhood search procedures (Yu et al., 2011). For the graph colouring problem (GCP), Fidanova and Pop (2016) developed an ACO and local search based algorithm, whilst Qin et al. (2011) opted for a combination of local search with a particle swarm optimisation procedure. Moalic and Gondran (2018) and Lü and Hao (2010) both presented memetic algorithms incorporating tabu search for the GCP, with the former using an EA comprising a population of just two solutions, and the latter implementing a novel multi-parent recombination operator.

[1] Our model, which is implemented in Xpress Mosel and uses FICO Xpress Optimizer, is available online (Hawa, 2020c).

An abundance of literature also exists on the topic of combined algorithms for packing problems. In 2010, Delorme et al. produced an algorithm using EA and GRASP characteristics for a generalisation of the set packing problem and, for the knapsack problem, Leung et al. (2012) developed a greedy algorithm which was then improved using simulated annealing. For the strip packing problem, proposed algorithms include a GRASP and variable neighbourhood search (VNS) approach (Beltrán et al., 2004), an ACO and basic placement heuristic with learning capabilities (Thiruvady et al., 2008), and a VNS procedure with improved constructive heuristics (Zhang et al., 2016). In addition, other unique combinations have been researched for the BPP. Hemmelmayr et al. (2012) decided on a VNS method which makes use of lower-bounding techniques and dynamic programming for the variable-sized BPP, whereas Kierkosz and Luczak (2014) opted for an EA which generates solutions to be used in a tree search improvement procedure for a generalised packing problem where the total area of the packing is to be maximised. Most notably, for the oriented two-dimensional BPP, Blum and Schmid (2013) proposed an EA featuring a randomised one-pass heuristic for constructing solutions which was able to find optimal solutions for four previously unsolved problem instances.

Due to the vast range of combined algorithmic approaches to combinatorial optimisation problems in literature, Talbi (2002) has published a taxonomy classifying such "hybrid" methods and provides a common terminology. For further information, the reader is directed to the work of Talbi (2013) which provides a complete background on the topic of combined metaheuristics as well as an abundance of examples and real-world applications.

6.2 An Exact Cover Formulation for the SCPP

Firstly, let us consider the following definition:

Definition 6.1 *Let $\mathcal{B}$ be a collection of subsets of a set $\mathcal{I}$. Then, a subcollection $\mathcal{S}^*$ of $\mathcal{B}$ is an* exact cover *if and only if each element in $\mathcal{I}$ is contained in exactly one subset in $\mathcal{S}^*$.*

For example, given a collection of subsets $\mathcal{B} = \{\{1,2,3\},\{3,4,5\},\{4\},\{5\}\}$ of a set $\mathcal{I} = \{1,2,3,4,5\}$, the subcollection $\mathcal{S}_1^* = \{\{1,2,3\},\{3,4,5\}\}$ is a *set cover* as it contains every element of $\mathcal{I}$ at least once, whereas the subcollection $\mathcal{S}_2^* = \{\{1,2,3\},\{4\},\{5\}\}$ is an *exact cover* as it contains every element of $\mathcal{I}$ exactly once. In this case, we say that each element in $\mathcal{I}$ is *covered* by exactly one subset in $\mathcal{S}_2^*$.

The exact cover problem, which seeks to determine whether an exact cover $\mathcal{S}^*$ of $\mathcal{B}$ exists, is a decision problem and one of Karp's 21 NP-complete problems (1972). Examples of exact cover problems include the popular puzzles Sudoku (Delahaye,

2006) and Pentomino tilings (Scott, 1958), whilst the n-queens problem can be seen as a generalised exact cover problem (Rivin et al., 1994).

If it is known that at least one exact cover exists for a given collection $\mathcal{B}$, then the exact cover problem can be transformed into an optimisation problem with the aim of finding the smallest exact cover in $\mathcal{B}$. Consider the following adaptation of the exact cover problem:

Definition 6.2 *Let $\mathcal{B}$ be a collection of subsets of a set $\mathcal{I}$. Then, the* Minimum Cardinality Exact Cover Problem (MXCP) *involves finding an exact cover $\mathcal{S}^* \subseteq \mathcal{B}$ such that $|\mathcal{S}^*|$ is minimised.*

Given a collection of subsets $\mathcal{B}$ of a set $\mathcal{I}$, let $b_{ij} = 1$ if element $i \in \mathcal{I}$ is in subset $j \in \mathcal{B}$. Then, the MXCP can be formulated as the following integer linear program:

$$\text{minimise} \quad \sum_{j \in \mathcal{B}} x_j \tag{6.2a}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{B}} b_{ij} x_j = 1 \qquad \forall\, i \in \mathcal{I} \tag{6.2b}$$

$$x_j \in \{0, 1\} \qquad \forall\, j \in \mathcal{B} \tag{6.2c}$$

$$x_j = \begin{cases} 1 & \text{if } j \in \mathcal{S}^* \\ 0 & \text{otherwise} \end{cases}$$

The MXCP can be directly linked to the SCPP where, given a collection $\mathcal{B}$ of feasible bins for an instance of the SCPP, the task is to determine the smallest subcollection of bins $\mathcal{S}^*$ that forms a feasible solution. Finding such an exact cover implies that each item in $\mathcal{I}$ is in exactly one bin in $\mathcal{S}^*$, fulfilling Conditions 2.7a and 2.7b of the SCPP. Indeed, since the bins are already feasible and meet Conditions 2.7c and 2.7d, the complications associated with the vicinal sum constraint are eliminated. A simple example is provided in Figure 6.2, where the collection $\mathcal{B}$ contains seven feasible bins for an instance of the SCPP. Although there exists an exact cover in $\mathcal{B}$ made up of five bins, the *minimum* cardinality exact cover $\mathcal{S}^*$ in $\mathcal{B}$ comprises four bins.

Recall that for the SCPP, $\mathcal{F}$ is the set of all possible feasible bins. Ideally, each element in the set $\mathcal{F}$ would correspond to a member of $\mathcal{B}$, however since the cardinality of $\mathcal{F}$ has the potential to grow at an exponential rate in relation to the problem size it may not be possible to produce $\mathcal{B}$ in this manner, let alone find an exact cover in a feasible amount of time. Nevertheless, it may be possible to identify methods that can produce a smaller, more manageable subset $\mathcal{B} \subset \mathcal{F}$ containing suitable bins. One option is to utilise a framework combining optimisation techniques called Generate and Solve, which we now consider.

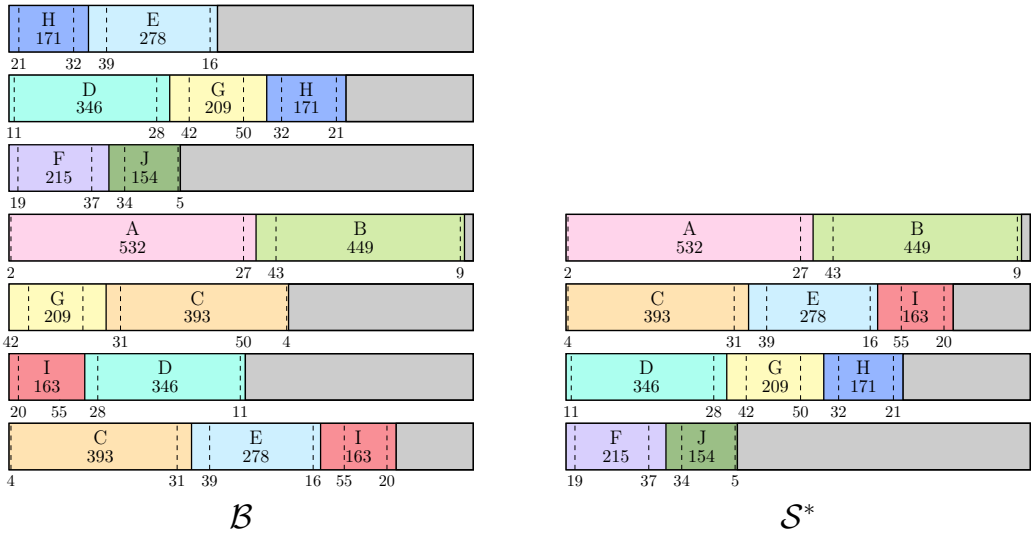


Figure 6.2: A collection $\mathcal{B}$ of feasible bins for an instance $\mathcal{I}$ of the SCPP, and a minimum cardinality exact cover $\mathcal{S}^* \subset \mathcal{B}$ comprising four bins. Here, $|\mathcal{I}| = 10$, $W = 1000$, and $\tau = 70$.

6.3 The Generate and Solve Framework

In the case of the SCPP, $\mathcal{F}$ comprises all feasible bins for the given instance $\mathcal{I}$ where each individual bin fulfils both the vicinal sum constraint and the bin capacity constraint. It then follows that all feasible solutions to the problem instance are in fact subsets of $\mathcal{F}$ that correspond to exact covers. As previously mentioned, $\mathcal{F}$ can become too large to completely enumerate in all but the most trivial of instances. Therefore, we can consider a subset $\mathcal{B} \subset \mathcal{F}$ containing a reduced collection of solution components from $\mathcal{F}$ for the problem instance that make up high quality solutions – a so-called "sub-instance". Then, an exact method could be applied to the more manageable sub-instance $\mathcal{B}$ to obtain high quality solutions to the original problem instance $\mathcal{I}$ with less computational effort and time in comparison to finding an optimal solution within the entire set $\mathcal{F}$ of feasible components.

This is the fundamental basis of a framework known as Generate and Solve (GS) (Nepomuceno et al., 2007a,b, 2008). The GS framework comprises two distinct modules. The first, called *Generator of Reduced Instances* (GRI), is responsible for producing components for the original problem instance that form a restricted problem instance $\mathcal{B}$. Any metaheuristic method can be implemented in the GRI provided the method creates components that satisfy the constraints of the original problem. For example, for the SCPP the GRI module would generate a number of feasible bins for the given problem instance. The second module is the *Solver of Reduced Instances* (SRI), which applies an exact method to find an optimal solution within the sub-instance $\mathcal{B}$ created by the GRI. An optimal solution to $\mathcal{B}$ will evidently also be a feasible solution to the original problem. In the case of the SCPP, this would

involve finding a minimum cardinality exact cover within the set of bins $\mathcal{B}$ created by the GRI. The objective function value of the optimal solution with respect to $\mathcal{B}$ is then used as feedback from the SRI module to the GRI module to guide the search process. This procedure is repeated until some stopping criteria is met, and the best solution obtained throughout the whole process is deemed to be the final solution to the original problem. Figure 6.3 illustrates the GS framework and process.

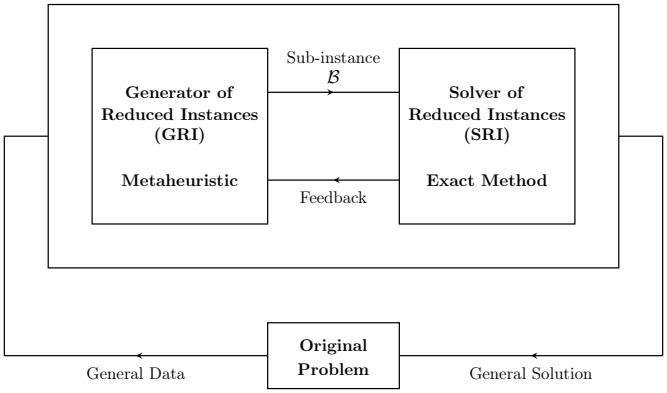


Figure 6.3: A diagram depicting the Generate and Solve framework, showing the relationship between the GRI and SRI modules.

This concept of reducing problem instances to solve to optimality has been exploited outside of the GS framework. One such example is that of Applegate et al. (1999) and Cook and Seymour (2003) for the TSP, where a metaheuristic is used to create high quality solutions that are then merged to produce a reduced problem instance. An exact solver is then used on the reduced instance to find an optimal solution. Klau et al. (2004) approached the Steiner tree problem in a similar manner, also incorporating a memetic algorithm based on a steady-state evolutionary algorithm. An ant colony optimisation algorithm is also suggested by Massen et al. (2012) and Massen et al. (2013) to construct a large number of feasible routes for the vehicle routing problem, from which a subset of routes is chosen using an exact solver for a relaxed set partitioning problem.

The GS framework has been utilised for a variety of optimisation problems. Procedures using a genetic algorithm in the GRI component have been developed for the container loading problem (Nepomuceno et al., 2007a,b; Saraiva et al., 2019) and cutting and packing problems (Nepomuceno et al., 2008; Pinheiro et al., 2011), which have also been adapted to include a simulated annealing method in place of the genetic algorithm (Saraiva et al., 2013). Similarly, algorithms based on the GS methodology have been constructed for wireless network efficiency problems using simulated annealing (Coudert et al., 2010, 2011) and genetic algorithms (Pinheiro et al., 2012). In each case, the problem has been mathematically formulated using integer programming for use with an exact solver.

6.4 The Construct, Merge, Solve & Adapt Algorithm

One particular instantiation of the GS framework is the Construct, Merge, Solve & Adapt (CMSA) algorithm of Blum et al. (2016). This is a generally applicable algorithm for combinatorial optimisation problems initially created for the minimum weight arborescence problem (Blum and Calvo, 2015).

The general CMSA algorithm, shown in Algorithm 12, operates as follows. The procedure begins with an empty sub-instance $\mathcal{B}$ which is to be filled with solution components, and an empty best-so-far solution $\mathcal{S}_{\text{bsf}}$ (Lines 1–2). Then, in each iteration, a fixed number q of solutions are *constructed* using the ConstructSolution function (Line 5). Each component of each solution is then *merged* into $\mathcal{B}$, provided the component does not already exist in $\mathcal{B}$, and the age of each new component, age[b], is initialised to zero (Lines 6–8). In the first iteration of CMSA, the best-so-far solution $\mathcal{S}_{\text{bsf}}$ is taken as the best of the q solutions created by the ConstructSolution function. Next, an exact *solver* is executed in the ApplyExactSolver function to find an optimal solution $\mathcal{S}^*$ to the sub-instance $\mathcal{B}$ (Line 11).

Algorithm 12 CMSA ($\mathcal{I}$, q, $\text{age}_{\max}$)

```
1:  B ← ∅
2:  S_bsf ← ∅
3:  while time limit not reached do
4:      for i ← 1 to q do
5:          S ← ConstructSolution()
6:          for all b ∈ S and b ∉ B do
7:              age[b] ← 0
8:              B ← B ∪ {b}
9:      if first iteration then
10:         S_bsf ← best solution S created by ConstructSolution()
11:     S* ← ApplyExactSolver(B)
12:     if S* is better than S_bsf then
13:         S_bsf ← S*
14:     for all b ∈ B do
15:         if b ∈ S* then
16:             age[b] ← 0
17:         else if b ∉ S* then
18:             age[b] ← age[b] + 1
19:             if age[b] = age_max then
20:                 B ← B\{b}
21: return S_bsf
```

If $\mathcal{S}^*$ is better than the current best-so-far solution $\mathcal{S}_{\text{bsf}}$, then solution $\mathcal{S}^*$ is stored as the new best-so-far solution (Lines 12–13). Finally, the sub-instance $\mathcal{B}$ is *adapted* using the solution $\mathcal{S}^*$ and an aging mechanism. The age of each component in $\mathcal{B}$ that is used in $\mathcal{S}^*$ is reset to zero, whilst the age of all other components in $\mathcal{B}$ is

increased by one (Lines 14–18). Any components in $\mathcal{B}$ whose age has reached the predefined maximum component age, $\text{age}_{\max}$, are removed from $\mathcal{B}$ (Lines 19–20).

The adaptation stage aids the process by removing components from $\mathcal{B}$ that have not contributed to an optimal solution for a set period of time. This allows the algorithm to control the size of $\mathcal{B}$ and therefore the speed of the exact solver. It also retains components in $\mathcal{B}$ that are used in solutions, which could prove to be useful in subsequent iterations of CMSA. Moreover, resetting the age of the components that are in the optimal solution to zero in each iteration ensures that there is at least one solution that can be formed by the components of the sub-instance $\mathcal{B}$ at all times.

This algorithm, which maintains and evolves a smaller, more manageable set of bins $\mathcal{B} \subset \mathcal{F}$, is a promising approach to the SCPP due to its focus on collecting high quality groups of bins, which has been seen to be a successful technique within our EA in the previous chapter by way of the GGA recombinaton operator.

The CMSA algorithm has been applied to a variety of combinatorial optimisation problems since its creation. Blum initially implemented the procedure on string problems, namely a generalisation of the minimum common string partition problem (Blum, 2016) and a special case of the longest common subsequence problem (Blum and Blesa, 2016), before exploring adaptations of CMSA involving the input format (Blum and Santos, 2019) and the feedback mechanism (Blum and Pereira, 2016), which was implemented on a multi-dimensional knapsack problem. A CMSA algorithm for the knapsack problem was also investigated by Lizárraga et al. (2017), comparing it against a large neighbourhood search (LNS) algorithm using varying problem sizes. Following on from the weighted arborescence problem, CMSA has been applied to a number of graph problems: the maximum happy vertices problem involving graph colouring (Lewis et al., 2019), the weighted independent domination problem (Pinacho-Davidson et al., 2018), as well as the minimum capacitated dominating set problem (Pinacho-Davidson et al., 2019). Other applications of CMSA include resource-constrained project scheduling (Thiruvady et al., 2019) and cooperative vehicle route planning with fuel constraints (Arora et al., 2019).

We now present two algorithms based on the CMSA methodology that have been modified appropriately for the SCPP, both of which also feature evolutionary characteristics. Note that in the general CMSA algorithm described above, and in the remainder of this chapter, the solution $\mathcal{S}^*$ is the optimal solution with respect to the sub-instance $\mathcal{B}$, as opposed to the optimal solution with respect to $\mathcal{F}$ which would be the global optimum to the problem instance overall.

6.4.1 CMSA with Mutation

Our first CMSA algorithm adapted for the SCPP is CMSA-M, which incorporates the mutation operator implemented in our EA from the previous chapter. The pseudocode for the ConstructSolution function used in CMSA-M is provided in Algorithm 13, which takes in $\mathcal{I}$ and $\mathcal{S}_{\text{bsf}}$ as arguments. In the first iteration of CMSA-M the sub-instance $\mathcal{B}$ is empty, so a fixed number q of initial solutions are generated using heuristics – one using MFFD$^+$ and the remaining $q-1$ using MFFR$^+$ – and each solution is mutated (Lines 1–4). In all subsequent iterations of CMSA-M, the incumbent best-so-far solution $\mathcal{S}_{\text{bsf}}$ is mutated q times to produce q new solutions (Line 5).

Algorithm 13 CMSA-M: ConstructSolution($\mathcal{I}$, $\mathcal{S}_{\text{bsf}}$)

1: **if** first iteration **then**
2: **if** $i = 1$ **then** $\mathcal{S} \leftarrow$ MFFD$^+(\mathcal{I})$
3: **else** $\mathcal{S} \leftarrow$ MFFR$^+(\mathcal{I})$
4: $\mathcal{S} \leftarrow$ Mutate($\mathcal{S}$)
5: **else** $\mathcal{S} \leftarrow$ Mutate($\mathcal{S}_{\text{bsf}}$)
6: **return** $\mathcal{S}$

This method of constructing solutions by mutating the incumbent best-so-far solution $\mathcal{S}_{\text{bsf}}$ may be beneficial as it encourages higher quality bins to be created and added to $\mathcal{B}$, which in turn increases the possibility of high quality solutions being found by the exact solver. On the other hand, modifying the same solution to create all subsequent solutions may prevent the inclusion of new, different solution components to the sub-instance $\mathcal{B}$, which could potentially limit exploration of the solution space.

6.4.2 CMSA with EA

Our second CMSA algorithm, CMSA-EA, implements our EA from the previous chapter in its entirety. This version begins by creating an initial population $\mathcal{P}$ of candidate solutions using the same method designed for our EA. Then, in each iteration of CMSA-EA, q new solutions are produced from the population $\mathcal{P}$ using recombination, mutation, and local search, as shown in Algorithm 14.

Note that rather than using just one recombination operator, the ConstructSolution function for CMSA-EA produces an offspring $\mathcal{S}$ using each of our three recombination operators – GGA, AGX, and AGX$'$ – from the same two parent solutions (Lines 5–8). The best offspring solution overall, $\mathcal{S}_{\text{best}}$, is returned by the function and also replaces the least fit of the parent solutions in $\mathcal{P}$ (Line 16).

As the offspring are inserted into the population $\mathcal{P}$ throughout the CMSA-EA algorithm, not only does the sub-instance $\mathcal{B}$ get updated but the population also

Algorithm 14 CMSA-EA: ConstructSolution($\mathcal{P}$)

```
select two parent solutions from P at random
S_best ← ∅
for i ← 1 to 3 do
    S ← ∅
    let S_1 and S_2 be copies of the original parent solutions
    if i = 1 then S ← GGA(S_1, S_2)
    else if i = 2 then S ← AGX(S_1, S_2)
    else if i = 3 then S ← AGX'(S_1, S_2)
    if ⋃_{S∈S} ≠ I then
        S' ← MFFD+(I\S)
        S ← LocalSearch(S, S')
    S ← Mutate(S)
    if i = 1 then S_best ← S
    else if S is better than S_best then
        S_best ← S
replace least fit parent solution with S_best in P
return S_best
```

evolves. Consequently, better parent solutions in $\mathcal{P}$ will create high quality offspring, which in turn results in superior bins being added to $\mathcal{B}$.

In each iteration of both CMSA-M and CMSA-EA the algorithms solve the sub-instance $\mathcal{B}$ by means of an exact technique using the ApplyExactSolver function. Finding a solution for the SCPP within the sub-instance $\mathcal{B}$ involves seeking a minimum cardinality exact cover; that is, solving the MXCP with respect to $\mathcal{B}$. To do so, we use a recursive depth-first backtracking algorithm process implemented using the "dancing links" technique of Knuth (2000). In general this algorithm, known as DLX, is designed to find all exact covers in a given sub-instance. However, as we are only interested in determining a single minimum cardinality exact cover $\mathcal{S}^* \subset \mathcal{B}$, i.e. a solution comprising the fewest bins, we modified this algorithm to create MinDLX, which only searches for exact covers that improve upon the best exact cover found so far.

Note also that our criteria for determining the "better" solution when comparing solutions in both CMSA-M and CMSA-EA algorithms is identical to that used for our EA: the solution comprising fewer bins is deemed the better solution, and if the solutions contain the same number of bins the one with the higher fitness value, calculated using (5.1), is taken as the better solution.

6.5 Computational Results

To compare the performance of our CMSA-M and CMSA-EA algorithms we randomly selected 50 problem instances from each of the 18 instance clases for the

SCPP. A time limit of 3600 seconds was also used for each of the algorithms, whilst MinDLX was set to run for a maximum of 600 seconds in each iteration. Parameter settings of $q = 3$ and $\text{age}_{\max} = 3$ were also decided after preliminary tests, and for CMSA-EA the population $\mathcal{P}$ was set to contain $|\mathcal{P}| = 25$ solutions, keeping in line with our previous EA experiments. The source code for both CMSA-M and CMSA-EA along with the MinDLX procedure is available online (Hawa, 2020d).[2] As with previous chapters, Tables 6.1 to 6.3 show the results obtained from our experiments using different values of δ.

In Table 6.1, for artificial instances we see that CMSA-EA produces solutons using the fewest number of bins $|\mathcal{S}|$ on average for larger item sets, whilst for the real instance classes CMSA-M is seen to be more suited for the two largest bin capacities. Due to the small value of $\delta = 0.25$ (and therefore a lower proportion of score widths meeting the vicinal sum constraint) we see that, similarly to previous experiments in Chapters 4 and 5, it is more difficult to find solutions comprising t bins. Consequently, there is only one instance class where the algorithms are able to find a solution for a single instance that contains t bins.

As expected, across all instance classes, the algorithm that creates the best solutions on average also produces more solutions that use fewer bins than the corresponding solution of the alternative algorithm. For three of the instance classes, CMSA-M produces solutions that use the same number or fewer bins that the CMSA-EA counterpart for all 50 instances, however there are no instance classes in which all solutions produced by CMSA-EA are better than or equal to the CMSA-M solutions.

Our experience with this algorithm also reveals a pattern with regards to the number of iterations of each method. For all instance classes where CMSA-M is superior, the average number of iterations performed by CMSA-M is greater than the average number of iterations of CMSA-EA. For example, for the real instance class using $W = 5000$ and $|\mathcal{I}| = 500$ the average number of iterations were 201 and 8 for CMSA-M and CMSA-EA respectively, whilst for artificial instances when $W = 1250$ and $|\mathcal{I}| = 100$ the corresponding values were 1681137 and 51. On the other hand, with the instance classes where the number of iterations of the algorithms are roughly equal, CMSA-EA is the method that produces the solutions using the fewest bins.

Turning to Table 6.2, the increase in δ is immediately apparent as we see that solutions comprising t bins have been found by at least one of the two algorithms for 13 of the 18 instance classes. In fact, for the artificial instance class when $W = 2500$ and $|\mathcal{I}| = 100$, all solutions produced by both CMSA-M and CMSA-EA use t

[2] As with our previous experiments, all experiments were implemented in C++ and executed on Windows machines with Intel Core i5-6500 3.20GHz processors and 8GB of RAM.

Table 6.1: Best solutions obtained from the CMSA-M and CMSA-EA algorithms for $\delta = 0.25$. Figures in bold indicate the best results for each instance class. Asterisks indicate statistical significance at $\leq 0.05(^*)$ and $\leq 0.01(^{**})$ according to a two-tailed paired t-test and two-tailed McNemar's test for the $|\mathcal{S}|$ and $\#t$ columns respectively.

$\delta = 0.25$			CMSA-M			CMSA-EA		
Type, W	$\lvert\mathcal{I}\rvert$	t^a	$\lvert\mathcal{S}\rvert^b$	$\#t^c$	Best^d	$\lvert\mathcal{S}\rvert$	$\#t$	Best
a, 1250	100	46.22	****54.10** $\pm$ 8.0	0	**20**	54.64 $\pm$ 7.7	0	0
	500	228.82	269.54 $\pm$ 2.3	0	3	****265.64** $\pm$ 2.1	0	**45**
	1000	459.94	540.66 $\pm$ 1.9	0	1	****532.94** $\pm$ 1.8	0	**48**
a, 2500	100	23.36	**36.94** $\pm$ 12.0	0	**7**	37.04 $\pm$ 12.3	0	3
	500	114.74	****190.20** $\pm$ 7.6	0	**35**	195.46 $\pm$ 4.3	0	12
	1000	230.24	397.40 $\pm$ 4.4	0	7	****387.50** $\pm$ 4.1	0	**42**
a, 5000	100	11.96	***36.68** $\pm$ 12.8	0	**6**	36.80 $\pm$ 12.6	0	0
	500	57.62	****176.00** $\pm$ 6.9	0	**43**	180.96 $\pm$ 5.0	0	5
	1000	115.42	362.06 $\pm$ 4.9	0	21	***359.30** $\pm$ 4.4	0	**27**
r, 1250	100	46.12	63.84 $\pm$ 15.1	0	1	**63.80** $\pm$ 15.1	0	**3**
	500	230.94	319.30 $\pm$ 13.0	0	3	****316.40** $\pm$ 13.2	0	**37**
	1000	461.96	638.96 $\pm$ 12.8	0	2	****632.50** $\pm$ 13.0	0	**35**
r, 2500	100	23.34	**46.64** $\pm$ 25.3	**1**	**5**	46.74 $\pm$ 25.0	**1**	3
	500	115.76	****234.44** $\pm$ 25.1	0	**30**	239.02 $\pm$ 22.8	0	11
	1000	231.26	485.00 $\pm$ 23.5	0	22	***479.30** $\pm$ 23.3	0	**23**
r, 5000	100	11.94	***45.84** $\pm$ 27.5	0	**5**	45.94 $\pm$ 27.2	0	0
	500	58.12	****228.08** $\pm$ 27.2	0	**31**	232.02 $\pm$ 25.6	0	8
	1000	115.92	**465.32** $\pm$ 27.1	0	**25**	466.50 $\pm$ 25.5	0	18

[a] $t = \lceil \sum_{i=1}^{n} w_i / W \rceil$ (mean from 50 instances).
[b] Number of bins per solution (mean from 50 instances plus or minus the coefficient of variation (%)).
[c] Number of instances in which the solution comprises t bins.
[d] Number of instances in which the solution comprises the fewest bins of the two algorithms.

bins, and CMSA-M also yields optimal solutions for all 50 artificial instances using $W = 5000$ and $|\mathcal{I}| = 500$.

Again, the same trend exists whereby CMSA-M produces solutions using fewer bins on average for the two largest bin capacities, i.e. when the number of items per bin is higher, whilst when using $W = 1250$ and the two larger item set sizes CMSA-EA creates better solutions for both real and artificial problem types. There are four instance classes where, although CMSA-M is able to produce solutions that use t bins, no such solutions are returned by CMSA-EA. Looking at the average number of iterations performed by each algorithm for these four classes, it is clear that the low number of CMSA-EA iterations, ranging on average between 7 and 8 in comparison to CMSA-M, which range from 49 to 84144, restricts the ability for the algorithm to seek superior solutions. Futhermore, there are seven classes for which all solutions returned by CMSA-M consist of the same number or fewer bins than the corresponding solutions produced by CMSA-EA and for all such classes the average number of CMSA-M iterations consistently exceeds the average number of CMSA-EA iterations.

Finally in Table 6.3, where $\delta = 0.75$, CMSA-M and CMSA-EA both produce solutions using the same number of bins on average for six of the 18 instance classes,

Table 6.2: Best solutions obtained from the CMSA-M and CMSA-EA algorithms for $\delta = 0.5$. Figures in bold and asterisks should be interpreted as in Table 6.1.

$\delta = 0.5$			CMSA-M			CMSA-EA		
Type, W	$\lvert\mathcal{I}\rvert$	t^a	$\lvert\mathcal{S}\rvert^b$	$\#t^c$	Best^d	$\lvert\mathcal{S}\rvert$	$\#t$	Best
a, 1250	100	46.22	***47.86** $\pm$ 7.4	**13**	**5**	47.96 $\pm$ 7.3	9	0
	500	228.82	233.82 $\pm$ 2.3	0	16	**233.74** $\pm$ 2.2	0	**21**
	1000	459.94	470.58 $\pm$ 1.7	0	17	**469.64** $\pm$ 1.5	0	**28**
a, 2500	100	23.36	**23.36** $\pm$ 4.5	**50**	0	**23.36** $\pm$ 4.5	**50**	0
	500	114.74	****114.94** $\pm$ 1.9	****41**	**50**	117.50 $\pm$ 2.0	0	0
	1000	230.24	****232.82** $\pm$ 1.5	**1**	**45**	235.86 $\pm$ 1.8	0	1
a, 5000	100	11.96	**12.26** $\pm$ 8.9	**43**	**1**	12.28 $\pm$ 9.5	**43**	0
	500	57.62	****57.62** $\pm$ 1.9	****50**	**27**	58.52 $\pm$ 2.5	23	0
	1000	115.42	****115.50** $\pm$ 1.4	****46**	**46**	119.48 $\pm$ 4.1	3	0
r, 1250	100	46.12	**53.64** $\pm$ 15.4	**1**	**5**	53.74 $\pm$ 15.3	0	2
	500	230.94	271.62 $\pm$ 13.2	0	5	****269.72** $\pm$ 13.4	0	**26**
	1000	461.96	545.54 $\pm$ 12.8	0	4	****540.48** $\pm$ 13.1	0	**28**
r, 2500	100	23.34	**26.56** $\pm$ 29.0	**33**	**6**	26.74 $\pm$ 2.9	31	1
	500	115.76	****134.72** $\pm$ 29.0	****15**	**46**	144.42 $\pm$ 25.9	0	3
	1000	231.26	****279.80** $\pm$ 28.4	0	**38**	290.62 $\pm$ 25.5	0	8
r, 5000	100	11.94	**19.96** $\pm$ 54.7	**19**	**2**	20.00 $\pm$ 54.5	18	0
	500	58.12	****92.86** $\pm$ 55.7	****21**	**41**	103.24 $\pm$ 50.2	7	0
	1000	115.92	****187.64** $\pm$ 55.8	****18**	**45**	214.98 $\pm$ 47.4	2	1

and for three of these classes all solutions comprise t bins. CMSA-EA also finds optimal solutions for all 50 artificial problem instances when $|\mathcal{I}| = 1000$ and $W = 5000$. The number of solutions found by the algorithms that consist of t bins is higher in accordance with the increase in the proportion of score widths that are able to fulfil the vicinal sum constraint; however there are still four instance classes – the artificial and real instance types using the two largest problem instance sizes and $W = 1250$ – for which neither of the CMSA algorithms yield a single solution using t bins.

Note that both CMSA-M and CMSA-EA performed fewer iterations on average in comparison to our previous EA experiments, despite the EA running for one sixth of the time period. The recombination operators in our EA require only two parent solutions from the population – which in the worst-case would each comprise $|\mathcal{I}|$ bins – and simply involves selecting bins from each parent solution for the offspring. The local search procedure applied on partial solutions is the most computationally expensive operator, however using the AHC algorithm aids the process. On the other hand, the MinDLX procedure is executed on the entire set of bins $\mathcal{B}$, which in some cases was seen to contain over 4000 bins. In preliminary trials, we allowed MinDLX to breach the 600 second time limit if no solution had been found, however this led to impractical run times. Therefore, in the event that MinDLX is unable to find a solution $\mathcal{S}^*$ with respect to $\mathcal{B}$ within the time limit, $\mathcal{S}^*$ is set to $\mathcal{S}_{\text{bsf}}$.

Enforcing the time limit in this manner caused issues in some of the tests. Obviously, as $|\mathcal{B}|$ increases, the MinDLX procedure will not be able to assess all combi-

Table 6.3: Best solutions obtained from the CMSA-M and CMSA-EA algorithms for $\delta = 0.75$. Figures in bold and asterisks should be interpreted as in Table 6.1.

$\delta = 0.75$			CMSA-M			CMSA-EA		
Type, W	$\lvert\mathcal{I}\rvert$	t^a	$\lvert\mathcal{S}\rvert^b$	$\#t^c$	Best^d	$\lvert\mathcal{S}\rvert$	$\#t$	Best
a, 1250	100	46.22	**47.68** ± 7.1	**15**	0	**47.68** ± 7.1	**15**	0
	500	228.82	231.86 ± 2.2	0	11	*__231.40__ ± 2.1	0	**18**
	1000	459.94	466.40 ± 1.5	0	13	**__464.00__ ± 1.4	0	**30**
a, 2500	100	23.36	**23.36** ± 4.5	**50**	0	**23.36** ± 4.5	**50**	0
	500	114.74	**__114.88__ ± 2.0	**__43__	**13**	115.14 ± 1.9	30	0
	1000	230.24	**__231.30__ ± 1.4	**4**	**11**	231.52 ± 1.4	0	0
a, 5000	100	11.96	**11.96** ± 5.3	**50**	0	**11.96** ± 5.3	**50**	0
	500	57.62	**57.62** ± 1.9	**50**	0	**57.62** ± 1.9	**50**	0
	1000	115.42	115.50 ± 1.4	46	0	*__115.42__ ± 1.4	**50**	**4**
r, 1250	100	46.12	**51.24** ± 15.7	**2**	**2**	51.26 ± 15.6	1	1
	500	230.94	258.44 ± 13.5	0	8	**257.98** ± 13.3	0	**15**
	1000	461.96	518.30 ± 13.1	0	6	**__516.44__ ± 13.1	0	**19**
r, 2500	100	23.34	**23.56** ± 9.6	43	**1**	**23.56** ± 9.6	**44**	**1**
	500	115.76	**__116.70__ ± 8.6	**__34__	**49**	119.84 ± 9.3	1	0
	1000	231.26	**__235.26__ ± 9.3	**1**	**47**	240.92 ± 9.5	0	2
r, 5000	100	11.94	**12.56** ± 22.0	**47**	0	**12.56** ± 22.0	**47**	0
	500	58.12	*__60.16__ ± 18.9	**__48__	**11**	61.00 ± 21.6	39	0
	1000	115.92	**__119.96__ ± 19.7	**__47__	**22**	124.00 ± 22.6	27	0

nations of the bins in $\mathcal{B}$ within the time limit and so may not be able to find the best solution $\mathcal{S}^*$, even if the appropriate bins are available in $\mathcal{B}$. Clearly, if MinDLX reaches the 600 second time limit in many iterations, the total number of CMSA iterations will be very low. Therefore, the solution returned by the algorithm will only be a result of these few iterations, and $\mathcal{B}$ will not have had the opportunity to evolve. That is, of course, assuming that a solution $\mathcal{S}^*$ is found in each iteration and is better than the incumbent $\mathcal{S}_{\text{bsf}}$. However, this is not always the case, and it may be that the solution returned by CMSA is simply the best-so-far solution set in the first iteration (see Lines 9–10 of Algorithm 12).

For example, consider the results produced by CMSA-EA for the real instance class when $|\mathcal{I}| = 100$, $W = 1250$, and $\delta = 0.25$. Of the 50 problem instances, 48 comprised just six iterations as MinDLX reached the time limit in each iteration, and two problem instances consisted of seven iterations, with MinDLX terminating in under 600 seconds in the first iteration, and reaching the time limit in the subsequent six iterations. There were only eight problem instances of the 48 where MinDLX found a solution $\mathcal{S}^*$ with respect to $\mathcal{B}$ in every iteration; the remaining 40 instances all contained at least one iteration where MinDLX was unable to return a solution. In fact, there were five problem instances whereby not a single solution $\mathcal{S}^*$ was found using MinDLX in all six iterations.

Furthermore, although MinDLX may find a solution $\mathcal{S}^*$ within the time limit, the solution may not be the *minimum cardinality* exact cover. For MinDLX to find the smallest exact cover, the procedure would have to be left to run so that all

possible bin combinations can be assessed, which as previously explained would take an unreasonable amount of time.

As briefly mentioned, fewer iterations of CMSA-EA were performed in comparison to CMSA-M in the majority of our experiments. One reason is due to the ConstructSolution function of each method. In CMSA-M, the construction of a single solution $\mathcal{S}$ simply requires using local search to mutate the incumbent best-so-far solution $\mathcal{S}_{\text{bsf}}$. In contrast, creating a single solution in CMSA-EA involves producing *three* individual offspring solutions using each of our recombination operators and executing the local search procedure up to two times for each offspring. However, although important to note, this would not make enough of an impact to contribute to the vast differences in the number of iterations. The main reason is not because of the time taken for the ConstructSolution function to produce solutions, but is a result of the actual solutions themselves. The solutions created by ConstructSolution in CMSA-M are created by mutating $\mathcal{S}_{\text{bsf}}$, so it is expected that some bins in the solution $\mathcal{S}$ will be identical to bins in $\mathcal{S}_{\text{bsf}}$ or bins that are present in $\mathcal{B}$; thus these bins will not be added to $\mathcal{B}$. Conversely, the solutions created by ConstructSolution in CMSA-EA are not derived from $\mathcal{S}_{\text{bsf}}$, and are less likely to contain as many duplicate bins. Consequently, the set $\mathcal{B}$ in CMSA-EA comprises a greater number of bins than in CMSA-M, and so MinDLX will tend to be slower.

Despite the computational issues observed, there are positive characteristics associated with our CMSA approaches. Using an exact solver is advantageous: MinDLX will, given a sufficient amount of time, produce a solution $\mathcal{S}^*$ that is optimal with respect to $\mathcal{B}$. On the other hand, although the recombination and local search stages of the EA perform much faster than MinDLX due to the input size, there is no way to determine whether the resulting offspring solution $\mathcal{S}$ generated using these operators is the best possible solution that can be produced from the two initial parent solutions. In CMSA, the entire set $\mathcal{B}$ is used in MinDLX and all bins in $\mathcal{B}$ are adapted accordingly. In other words, every bin in $\mathcal{B}$ serves a purpose and is utilised in every iteration. Conversely, the EA selects just two solutions from the population $\mathcal{P}$, disregarding the remaining solutions, and only one solution is replaced in $\mathcal{P}$ in each iteration. Moreover, our CMSA-M algorithm has a feedback property, which makes use of the incumbent best-so-far solution $\mathcal{S}_{\text{bsf}}$ to assist the construction of new bins in subsequent iterations. Note also that the set $\mathcal{B}$ does not permit duplicate bins, whilst in the EA there is the possibility of identical bins appearing in multiple solutions in the population, or even entirely identical solutions.

6.6 Summary

In this chapter we turned our attention to exact methods for the SCPP. We began by formulating an ILP model for a generalised vehicle routing problem that corresponds to the SCPP and explained that, as the SCPP is NP-hard, the model is only useful for finding optimal solutions for very small problem instances. From this, we introduced the Minimum Cardinality Exact Cover Problem (MXCP) and discussed how exact methods can be integrated into wider optimisation frameworks such as Generate and Solve (GS). After reviewing the Construct, Merge, Solve & Adapt (CMSA) algorithm, an instantiation of GS which operates on a population of individual bins rather than entire solutions, we presented two custom CMSA-based algorithms for the SCPP: CMSA-M, which adopts a mutation operator to create new solutions from the current best-so-far solution $\mathcal{S}_{\text{bsf}}$, and CMSA-EA where each new solution is produced using a single iteration of our EA.

Our initial intention was to implement a post-optimisation stage in a similar manner to Malaguti et al. (2008), by copying the bins of each offspring solution created by our EA into a set $\mathcal{B}$ and then simply applying an exact solver to obtain a minimum cardinality exact cover with respect to $\mathcal{B}$. However, as explained in the previous chapter, the size of the set of offspring bins became too large to maintain. Therefore, we developed the CMSA-EA algorithm as an alternative method that uses a smaller number of bins from offspring solutions in each iteration.

Both CMSA algorithms were hindered by MinDLX, the procedure implemented to solve the MXCP with respect to $\mathcal{B}$. Although in our experiments we set $q = 3$, $\mathcal{B}$ still grew to contain a great number of bins, particularly for larger problem instances when $W = 1250$. As a result, for many of the instances MinDLX struggled to find a single exact cover, let alone a minimum cardinality exact cover, within the specified time limit. One possible improvement would be to somehow limit the size of $\mathcal{B}$ – whilst also ensuring that at least one exact cover exists in $\mathcal{B}$ – so that a solution $\mathcal{S}^*$ can be found in a shorter period of time which would subsequently increase the number of overall CMSA iterations and encourage the evolution of bins in $\mathcal{B}$.

Another potential modification to consider is implementing a feedback mechanism within CMSA-EA. Currently, each solution $\mathcal{S}^*$ found by MinDLX in CMSA-EA is only used within the adapt section of the overall algorithm, and of course replaces the best-so-far solution $\mathcal{S}_{\text{bsf}}$ if it is of higher quality. The population $\mathcal{P}$ can thus be seen as a separate entity, where only the offspring solutions produced in ConstructSolution facilitate the evolution of $\mathcal{P}$. An improvement on CMSA-EA would involve incorporating $\mathcal{S}_{\text{bsf}}$ into the population $\mathcal{P}$, perhaps by replacing the least fit solution in $\mathcal{P}$ with $\mathcal{S}_{\text{bsf}}$ whenever a new best-so-far solution is found.

Chapter 7

An Alternative Version of the Score-Constrained Packing Problem

Recall that in the SCPP, items are to be packed alongside one another into the fewest number of bins such that no bin is overfilled and the vicinal sum constraint is fulfilled in every bin (see Definition 1.1). We have acknowledged that, due to the vicinal sum constraint, there can exist instances where no two score widths meet the vicinal sum constraint and so each item in $\mathcal{I}$ must be packed into individual bins; thus, a feasible solution $\mathcal{S}$ will comprise $|\mathcal{I}|$ bins.

Suppose, however, that we were to permit spaces between successive items in bins. Then, if the score widths of two adjacent items do not satisfy the vicinal sum constraint the items can be moved further apart from one another in the bin, creating a "gap" between the items so that the total distance between the two score lines is greater than or equal to the minimum scoring distance τ. This then allows the knives to be able to score the items correctly, as shown in Figure 7.1.

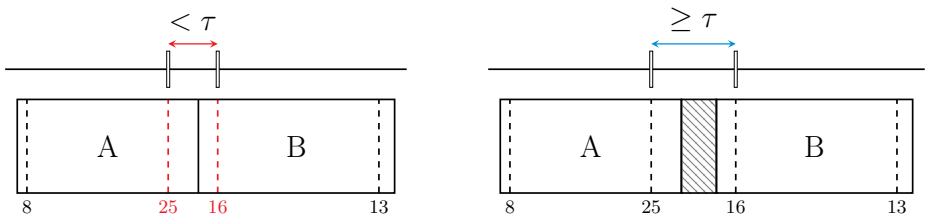


Figure 7.1: An example showing two items whose adjacent score widths do not total the minimum scoring distance, $\tau = 70$. By separating the items to create a space, indicated by the lined area, the distance between the neighbouring score lines of the items increases, allowing the knives to score along the items in the correct locations.

In this short final chapter we introduce an alternative version of the SCPP and discuss the differences between the two problems as well as potential approaches.

7.1 Definitions

Given a set $\mathcal{I}$ of n items and a minimum scoring distance $\tau \in \mathbb{Z}^+$, consider the following *modified* vicinal sum constraint:

$$\mathbf{r}(i) + \mathbf{l}(i+1) + \alpha(\mathbf{r}(i), \mathbf{l}(i+1)) \geq \tau \quad \forall\, i \in \{1, 2, \ldots, |S| - 1\}, \tag{7.1}$$

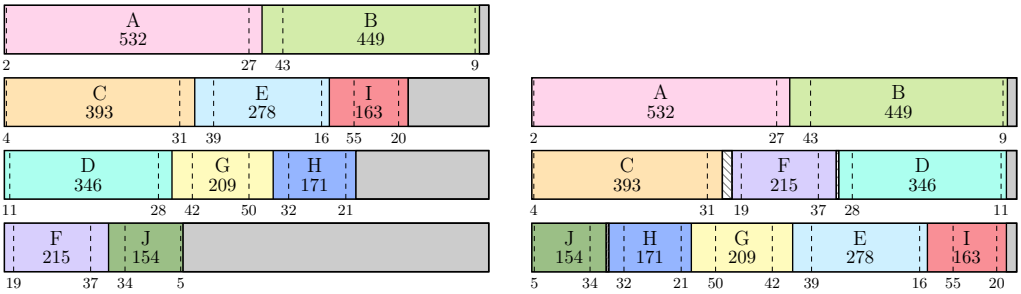
where $\mathbf{l}(i)$ and $\mathbf{r}(i)$ denote the left- and right-hand score widths of the ith item in the bin S, and $\alpha(\mathbf{r}(i), \mathbf{l}(i+1))$ denotes the *inter-item width* between the two successive items in the bin. The inter-item width is calculated as follows:

$$\alpha(\mathbf{r}(i), \mathbf{l}(i+1)) = \begin{cases} \tau - (\mathbf{r}(i) + \mathbf{l}(i+1)) & \text{if } \mathbf{r}(i) + \mathbf{l}(i+1) < \tau \\ 0 & \text{otherwise.} \end{cases}$$

From this, we introduce the Modified Score-Constrained Packing Problem:

Definition 7.1 *Let $\mathcal{I}$ be a set of n rectangular items of height $H > 0$ with varying widths $w_i \in \mathbb{Z}^+$ and score widths $a_i, b_i \in \mathbb{Z}^+$ for all $i \in \mathcal{I}$. Given a minimum scoring distance $\tau \in \mathbb{Z}^+$, the* Modified Score-Constrained Packing Problem (MSCPP) *involves packing the items from left to right into the fewest number of $H \times W$ bins such that the modified vicinal sum constraint is satisfied in each bin and no bin is overfilled.*

Figure 7.2 compares solutions to the original SCPP and the MSCPP for the same instance $\mathcal{I}$. It can be seen that permitting the use of space between items in bins (as with the MSCPP) allows more items to be packed together into each bin, resulting in fewer bins in this case.



(a) SCPP (b) MSCPP

Figure 7.2: A comparison of solutions for the SCPP and MSCPP using the same instance $\mathcal{I}$ where, due to the modified vicinal sum constraint, the solution for the MSCPP comprises fewer bins. The lined area between items in the MSCPP solution indicate the inter-item widths. Here, $|\mathcal{I}| = 10$, $W = 1000$, and $\tau = 70$.

In order for each bin to be packed properly and not be overfilled, not only must the total width of the items in each bin be less than or equal to the bin capacity, but

the inter-item widths must also be taken into consideration. So, in a similar manner to the SCPP, the MSCPP involves determining which items should be packed into each bin and how each item should be packed within each bin. This gives rise to the following sub-problem that occurs within the MSCPP.

Definition 7.2 *Let $\mathcal{I}' \subset \mathcal{I}$ be a subset of rectangular items whose total width is less than or equal to the bin capacity, i.e. $A(\mathcal{I}') = \sum_{i \in \mathcal{I}'} w_i \leq W$. Then, given a minimum scoring distance $\tau \in \mathbb{Z}^+$, the* Modified Score-Constrained Packing Sub-Problem (sub-MSCPP) *consists in finding an order and orientation of the items in $\mathcal{I}'$ into a bin S such that the modified vicinal sum constraint is fulfilled and the total inter-item width*

$$f(S) = \sum_{i=1}^{|S|-1} \alpha(\mathbf{r}(i), \mathbf{l}(i+1)) \tag{7.2}$$

is less than or equal to $W - A(\mathcal{I}')$.

Consider Figure 7.3, where a set $\mathcal{I}'$ of four items of total width $A(\mathcal{I}') \leq W$ are packed into two bins S_1 and S_2 in different configurations. Although the modified vicinal sum constraint is fulfilled in both bins, the arrangement of items in the first bin has a larger total inter-item width $f(S_1)$ which is greater than $W - A(\mathcal{I}')$, causing the bin to be overfilled. By changing the order and orientation of the items we obtain the packing shown in bin S_2, where $f(S_2) + A(\mathcal{I}') \leq W$.

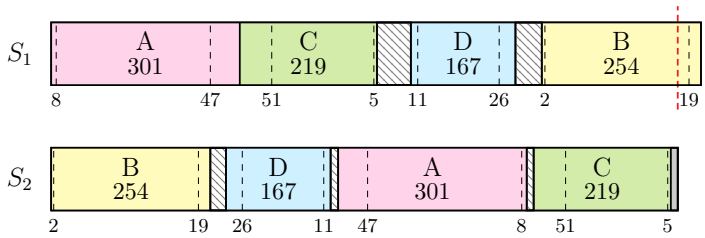


Figure 7.3: A set $\mathcal{I}'$ of four items packed into two bins of capacity $W = 1000$ in different arrangements, where $A(\mathcal{I}') = 941$ and $\tau = 70$. In bin S_1 the total inter-item width, $f(S_2) = 96$, causes the bin to be overfilled, whilst the alignment of items in bins S_2 has smaller total inter-item width $f(S_2) = 48$ and so can be packed into the bin feasibly. The red vertical dashed line on bin S_1 indicates the end of the bin.

Observe that, for any instance $\mathcal{I}'$ and minimum scoring distance τ, if there exists a feasible solution that fulfils the constraints of the sub-SCPP, then it is guaranteed that there also exists a feasible solution that satisfies the sub-MSCPP; however, the converse is not true. For any given instance, the set of all feasible bins $\mathcal{F}$ will be larger for the sub-MSCPP. Permitting spaces between items also means that removing an item from a feasible bin for the MSCPP maintains feasibility, unlike with the SCPP.

The MSCPP is similar to the cutting stock problem with sequence-dependent cut losses (CSP-SDCL) of Garraffa et al. (2016) which we reviewed in Chapter 2, where the inter-item widths in the MSCPP can be viewed as cut losses in the CSP-SDCL.

The cut losses between each pair of items in the CSP-SDCL is only dependent on the order of the items, however, whilst both the ordering and orientation of the items in the MSCPP affect the inter-item widths between items due to the modified vicinal sum constraint. It is also important to note the symmetry that occurs in the MSCPP: the inter-item width between two items A and B packed in regular orientations will be the same as the inter-item width between the items in their rotated orientations with B packed before A. On the other hand, recall that the cut losses between each pair of items in the CSP-SDCL are not symmetrical, and are also highly varied.

7.2 The sub-MSCPP

Although both the sub-SCPP and the sub-MSCPP involve packing a set $\mathcal{I}$ of n items with score widths into bins, the two problems differ in a number of ways. Firstly, consider the total width of the set of items $A(\mathcal{I})$: in the sub-SCPP, if $A(\mathcal{I}) \leq W$ then any feasible arrangement of the items will fit into a single bin S. However, in the sub-MSCPP there is no guarantee that a feasible configuration of a set of items with total width $A(\mathcal{I}) \leq W$ will fit into a bin S as the total inter-item width $f(S)$ may exceed $W - A(\mathcal{I})$. Secondly, finding a solution for the sub-SCPP involves seeking a single arrangement of the items such that the vicinal sum constraint is fulfilled. For an instance $\mathcal{I}$ of the sub-MSCPP, all $2^{n-1}n!$ distinct configurations of the n items are feasible with regards to the modified vicinal sum constraint (7.1), and so the task instead is to find the arrangement with minimum $f(S)$. Therefore, if the best configuration found has $f(S) > W - A(\mathcal{I})$ then it can be said for certain that there does not exist an arrangement of the items in $\mathcal{I}$ that can be packed into a single bin.

The sub-MSCPP can be presented as a generalised travelling salesman problem (TSP), where a salesman is to start from city 0 and visit n cities before returning back to city 0. The distance between two cities i and j is denoted by w_{ij} for all $i, j = 0, 1, \ldots, n, i \neq j$, and each city i is paired with one other city, $p(i)$, for $i = 1, \ldots, n$. The salesman is required to visit paired cities in succession. Thus, the problem of finding the shortest tour such that each city is visited exactly once can be formulated as the following ILP:

$$\text{minimise} \quad \sum_{i=0}^{n} \sum_{j=0}^{n} w_{ij} x_{ij} \tag{7.3a}$$

$$\text{subject to} \quad \sum_{i=0}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n \tag{7.3b}$$

$$\sum_{j=0}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{7.3c}$$

$$x_{i,p(i)} + x_{p(i),i} = 1 \qquad i = 1, \ldots, n \tag{7.3d}$$

$$u_0 = 1 \tag{7.3e}$$

$$u_i - u_j + 1 \leq (n-1)(1 - x_{ij}) \qquad i, j = 1, \ldots, n \tag{7.3f}$$

$$1 \leq u_i \leq n \qquad i = 1, \ldots, n \tag{7.3g}$$

$$u_i \in \mathbb{Z} \qquad i = 0, \ldots, n \tag{7.3h}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 0, \ldots, n \tag{7.3i}$$

$$x_{ij} = \begin{cases} 1 & \text{if the salesman travels from city } i \text{ to city } j \\ 0 & \text{otherwise.} \end{cases}$$

The objective function (7.3a) is to minimise the total distance of the tour travelled by the salesman. Constraints (7.3b) and (7.3c) state that each city must be visited exactly once, and the requirement for each city i, except city 0, to be preceeded or succeeded by its paired city in the tour is covered by Constraint (7.3d). Constraints (7.3e) – (7.3h) ensure that the tour begins and ends at city 0 and prevents subtours, and finally Constraint (7.3i) restricts the decision variable x_{ij} to integers 0 and 1. This ILP model, implemented in Xpress Mosel, is available online (Hawa, 2020b).

In this formulation, it can be seen that each city $i = 1, \ldots, n$ represents a score width of an item in the set $\mathcal{I}$, and paired cities correspond to score widths that are on the same item. Then, the distances w_{ij} denote either the item widths if the cities are paired or the inter-item widths. Similarly to our ILP for the SCPP in the previous chapter, this generalised TSP is NP-hard; finding an exact solution using a commercial solver for larger instances of the sub-MSCPP is not feasible. Therefore, it is worthwhile seeking alternative methods for finding arrangements of items with minimum total inter-item width.

7.2.1 An Algorithm for the sub-MSCPP

Due to the resemblances between the sub-SCPP and the sub-MSCPP, it would be beneficial to approach the latter in a manner similar to that of the former, as seen in Chapter 3. As before, we can model an instance $\mathcal{I}$ of the sub-MSCPP graphically by assigning two vertices u and v for every item $i \in \mathcal{I}$, with weights $w(u) = a_i$ and $w(v) = b_i$, along with a blue edge $\{u, v\}$. For parity with the sub-SCPP model we refer to these pairs of vertices as partners, and without loss of generality assume the vertices $\{v_1, \ldots, v_{2n}\}$ are indexed in order of non-decreasing weight, i.e. $w(v_i) \leq w(v_{i+1})$. An additional pair of partner vertices, v_{2n+1} and v_{2n+2}, each of weight τ,

is also introduced together with a blue edge $\{v_{2n+1}, v_{2n+2}\}$. The blue edges between partners form the edge set B, where the partner of each vertex v_i is denoted by $p(v_i)$. The graph G thus consists of a vertex set V of $2n+2$ vertices and an edge set B of $n+1$ edges (see Section 3.2 of Chapter 3).

In the sub-SCPP model, the set of edges R only contains edges between vertices whose corresponding score widths meet the vicinal sum constraint. This is where the two models differ: for the sub-MSCPP, the set R of red edges added to the graph G comprises $2n(n+1)$ edges where every vertex is adjacent to every other vertex via an edge in R, provided the vertices are not partners; that is, $R = \{\{v_i, v_j\} : p(v_i) \neq v_j \ \forall \ v_i, v_j \in V, v_i \neq v_j\}$. This is due to the modified vicinal sum constraint, as every pair of items in $\mathcal{I}$ can be placed alongside one another so long as the total distance between the neighbouring score lines is greater than or equal to the minimum scoring distance τ. Therefore, the edges in both B and R are weighted to account for the items' widths and inter-item widths. Recall that each vertex $v_i \in V$ represents a score width and each item $i \in \mathcal{I}$ can be denoted by its score widths, (a_i, b_i). Then, the edges in B and R are weighted as follows:

$$w(\{v_i, v_j\}) = \begin{cases} w_{(a_i,b_i)} & \text{if } \{v_i, v_j\} \in B \\ \tau - (w(v_i) + w(v_j)) & \text{if } w(v_i) + w(v_j) < \tau \text{ and } \{v_i, v_j\} \in R \\ 0 & \text{otherwise.} \end{cases}$$

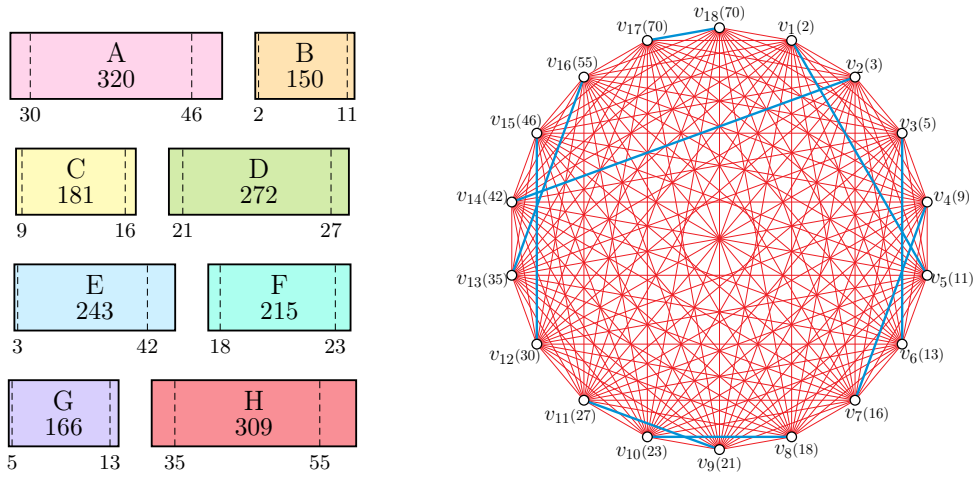
Note that because the additional vertices v_{2n+1} and v_{2n+2} do not represent an item, the weight of the edge $\{v_{2n+1}, v_{2n+2}\}$ is zero. Thus, the sum of all edges in B is equal to the total width of all items in $\mathcal{I}$:

$$w(B) = \sum_{\{v_i, p(v_i)\} \in B} w(\{v_i, p(v_i)\}) = A(\mathcal{I}).$$

The resulting graph $G = (V, B \cup R)$ is a vertex-weighted, edge-weighted complete graph. Figure 7.4 illustrates an example instance $\mathcal{I}$ of the sub-MSCPP and the corresponding graph G, where $|\mathcal{I}| = 8$, $\tau = 70$, and $w(B) = 1856$.

As with the original sub-SCPP, the task involves finding an alternating Hamiltonian cycle in G; however, in the case of the sub-MSCPP it is known that there exists $2^{n-1}n!$ such cycles in G. Instead, the aim is to obtain the shortest alternating Hamiltonian cycle. All edges in B will be present in the final cycle, and so the focus is on selecting edges from R of minimum total weight that form an alternating Hamiltonian cycle with the edges in B. To do this, we make use of the original AHC algorithm from Chapter 3, modifying the procedure to suit the sub-MSCPP.

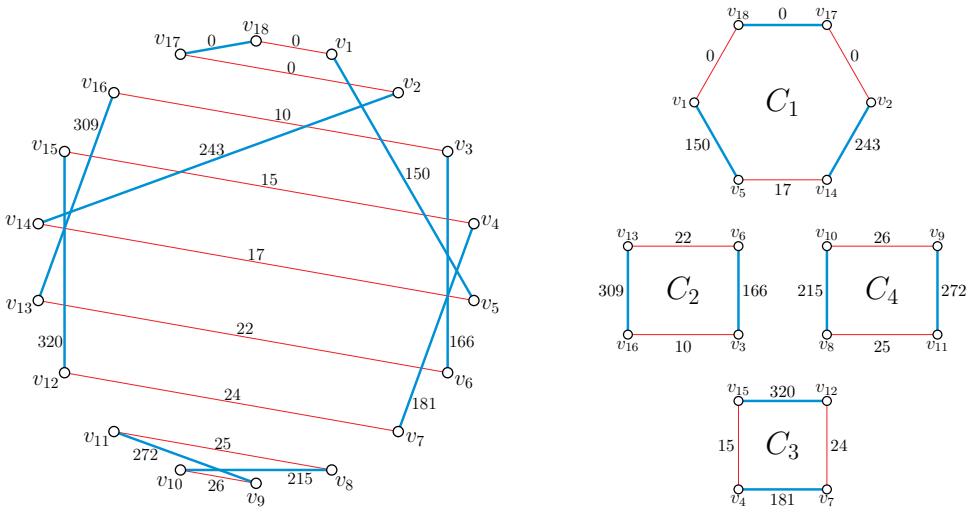
The first stage is to obtain a subset $R' \subset R$ of $n+1$ edges. The Maximum Cardinality Matching (MCM) algorithm used in AHC is suitable for this model,



(a) Problem instance $\mathcal{I}$ (b) $G = (V, B \cup R)$

Figure 7.4: (a) An instance $\mathcal{I}$ of the sub-MSPP comprising eight items; and (b) the graph $G = (V, B \cup R)$ modelling the instance $\mathcal{I}$, where the thicker blue edges are in B and the thinner red edges are in R, with the vertices' weights stated in parentheses. In this instance, $A(\mathcal{I}) = w(B) = 1856$ and $\tau = 70$. Note that due to the number of edges on G the edge weights are not labelled.

as it selects edges from R connecting the lowest-indexed vertices to the highest-indexed vertices, which will have minimum weight overall (see Algorithm 2). Note that, unlike with the sub-SCPP, there will always exist a set $R' \subset R$ such that $|R'| = n + 1$. The subgraph $G' = (V, B \cup R')$ then consists of cyclic components $C_1, \ldots, C_z$. Figure 7.5 shows the subgraph G' after MCM has been executed on our example instance, where it can be seen that G' consists of $z = 4$ components. In this case, the sum of the weights of edges in R' is $w(R') = 139$, and so the total length of G' is $w(B) + w(R') = 1995$.



(a) $G' = (V, B \cup R')$ (b) G' in planar form

Figure 7.5: The subgraph $G' = (V, B \cup R)$ using the edge set R' produced using MCM, where in planar form it can be seen that G' comprises $z = 4$ cyclic components. The total sum of edge weights in R' is $w(R') = 139$; thus the length of G' is $w(B) + w(R') = 1995$.

We are now able to provide a lower bound for the shortest length of an alternating Hamiltonian cycle in G, as stated in the following theorem.

Theorem 7.3 *Let $G = (V, B \cup R)$ be a vertex-weighted, edge-weighted graph modelling an instance $\mathcal{I}$ of the sub-MSCPP, and let $R' \subset R$ be the subset of edges procured by MCM. Then, the length of the shortest alternating Hamiltonian cycle in G is greater than or equal to $w(B) + w(R')$.*

Proof. As all edges in B must be present in the final alternating Hamiltonian cycle, the weights of the edges in B will contribute to the length of the cycle. Therefore, it must be shown that there does not exist any other matching in R that has total weight less than $w(R')$.

Recall that the vertices are labelled such that $w(v_{i+1}) \geq w(v_i)$ for all $v_i \in V$. Then, consider the vertices v_i, v_j, v_k, and v_l, where $i < j < k < l$. Given the ordering of the vertices, it follows that $w(\{v_i, v_l\}) + w(\{v_j, v_k\}) \leq w(\{v_i, v_k\}) + w(\{v_j, v_l\})$; thus the process of matching the smallest vertices with the largest vertices as with the MCM algorithm produces a matching R' of minimum weight. □

If G' comprises just one component then an alternating Hamiltonian cycle of minimum length has been found; else, the next stage is to remove edges from R and replace them with edges from $R \backslash R'$ that connect the multiple components into a single cycle. In the sub-SCPP, it is not known whether it is possible to merge the components together, and there may not exist an alternating Hamiltonian cycle. However, in the case of the sub-MSCPP there are a multitude of ways of joining the components as the underlying graph G is complete. Thus, the aim is to determine which edges to remove from R' and which edges to add to R' from $R \backslash R'$ such that the overall length of the resulting alternating Hamiltonian cycle is minimised.

For this, we implement a variant of the Bridge-Cover Recognition (BCR) procedure used in our original AHC algorithm, BCR′, that has been appropriately modified for the sub-MSCPP. The first part of this procedure is shown in Algorithm 15. To begin, the edges in R' are sorted into a list $\mathcal{L}$ such that the lower-indexed vertices of the edges are in increasing order. Then, BCR′ starts from the beginning of $\mathcal{L}$ in search of two successive edges that are in different components of G' (Lines 7–9).[1] The second of the two edges is referred to as the "starting edge" (Line 10). BCR′ adds these two edges to a subset R''_i (Lines 11–15) before proceeding to the next edge in $\mathcal{L}$, checking whether the edge is in a different component of G' to the edges in R''_i. If so, the edge is added to a *new* subset R''_{i+1} along with the edges in R''_i

[1] Note that BCR′ does not need to check that the lower-indexed vertex of the current edge is adjacent to the higher-indexed vertex of the next edge as with the original algorithm as G is a complete graph.

(Lines 16–20). This process continues until the next edge is in the same component as one of the previous edges. BCR′ then returns to the starting edge in the list and repeats the search for new subsets, terminating once the penultimate edge in $\mathcal{L}$ has been assessed (Line 21).

Algorithm 15 BCR′ ($G' = (V, B \cup R')$)

```
 1: create list L of edges in R'
 2: let k denote edges {v_j, m(v_j)} in L
 3: let l denote the last edge in L, and let l − 1 denote the penultimate edge in L
 4: startingEdge ← {v_1, m(v_1)}
 5: i ← 1
 6: repeat
 7:     k ← startingEdge
 8:     while k ≠ l − 1 and (k ∈ C_p and k + 1 ∈ C_p) do
 9:         k ← k + 1
10:     startingEdge ← k + 1
11:     if k ∈ C_p and k + 1 ∈ C_q : p ≠ q then
12:         R''_i ← {{k}, {k + 1}}
13:         S''_i ← {p, q}
14:         k ← k + 1
15:         i ← i + 1
16:         while k ≠ l and k + 1 ∈ C_r : r ∉ S''_{i−1} do
17:             k ← k + 1
18:             R''_i ← R''_{i−1} ∪ {{k}}
19:             S''_i ← S''_{i−1} ∪ {r}
20:             i ← i + 1
21: until startingEdge = l
```

It should be noted that all edges in R' are included in the list $\mathcal{L}$. In the original BCR procedure, the list $\mathcal{L}$ is sorted such that the lower-indexed vertices of the edges are in increasing order and the higher-indexed vertices of the edges are in decreasing order, and any edges that do not adhere to this criteria are removed. This is to ensure that the lower-indexed vertex of any edge in $\mathcal{L}$ (excluding the first edge) will be adjacent to the higher-indexed vertex of the previous edge in the list. However, in this model for the sub-MSCPP all vertices are adjacent to one another, and so there is no need to disregard any of the edges.

Once all possible subsets of edges have been created, the next stage is to find a collection of the subsets that can connect the cyclic components of G' together into a single alternating Hamiltonian cycle, as described in Section 3.3.2 of Chapter 3. Rather than attempting to construct just one collection $\mathcal{R}''$, however, BCR′ proceeds to assemble all possible collections $\mathcal{R}''_i$ of subsets. To distinguish from the initial matching R', let R'_M denote the modified version of the edge set that is produced by a collection $\mathcal{R}''_i$, where edges in R' have been removed and replaced. BCR′ calculates the sum of the edge weights $w(R'_M)$ for each collection, and the collection that yields the modified edge set of minimum weight is used to merge the components of G'

together. Note, however, that if BCR′ finds a collection that produces a modified edge set R'_M of the same total weight as R', i.e. $w(R'_M) = w(R')$, then on the basis of Theorem 7.3 no other collection exists that creates a shorter cycle; thus BCR′ concludes the search for further collections.

Figure 7.6 demonstrates the BCR′ procedure on our example instance, where 12 subsets have been obtained from the list $\mathcal{L}$. Starting from R''_1, BCR′ forms collections and calculates the sum of the edge weights of R'_M that each collection generates. As the set R'_M created by $\mathcal{R}''_4$ has the same total weight as R', BCR′ uses $\mathcal{R}''_4$ to procure the edges from $R \backslash R'$ that will replace edges in R'. Figure 7.7 shows the new modified set R' connecting the four cyclic components of G' into a single alternating Hamiltonian cycle. Removing the two additional vertices v_{2n+1} and v_{2n+2} and incident edges from the cycle forms a weighted alternating path that corresponds to an alignment of the items in $\mathcal{I}$ with inter-item widths, as shown in Figure 7.7c.

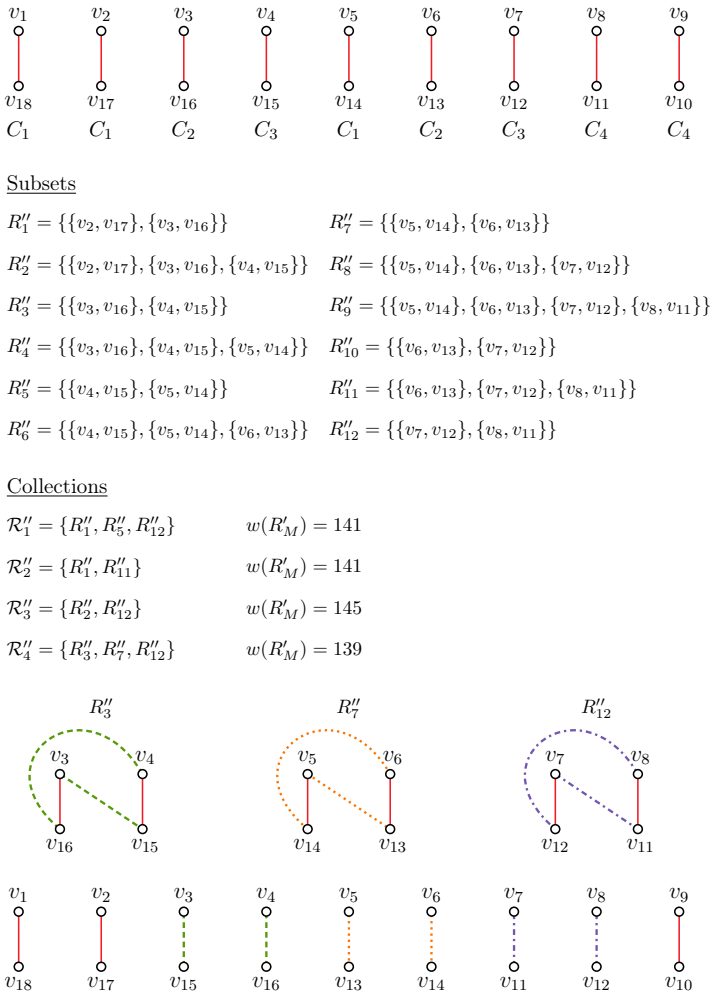


Figure 7.6: The BCR′ procedure operating on our example instance of the sub-MSCPP, where subsets $R''_1, \ldots, R''_{12}$ have been created from the list $\mathcal{L}$ of edges from R'. The collection $\mathcal{R}''_4$ produces a modified set R'_M of the same total weight as R'; thus $\mathcal{R}''_4$ is used to obtain edges from $R \backslash R'$.

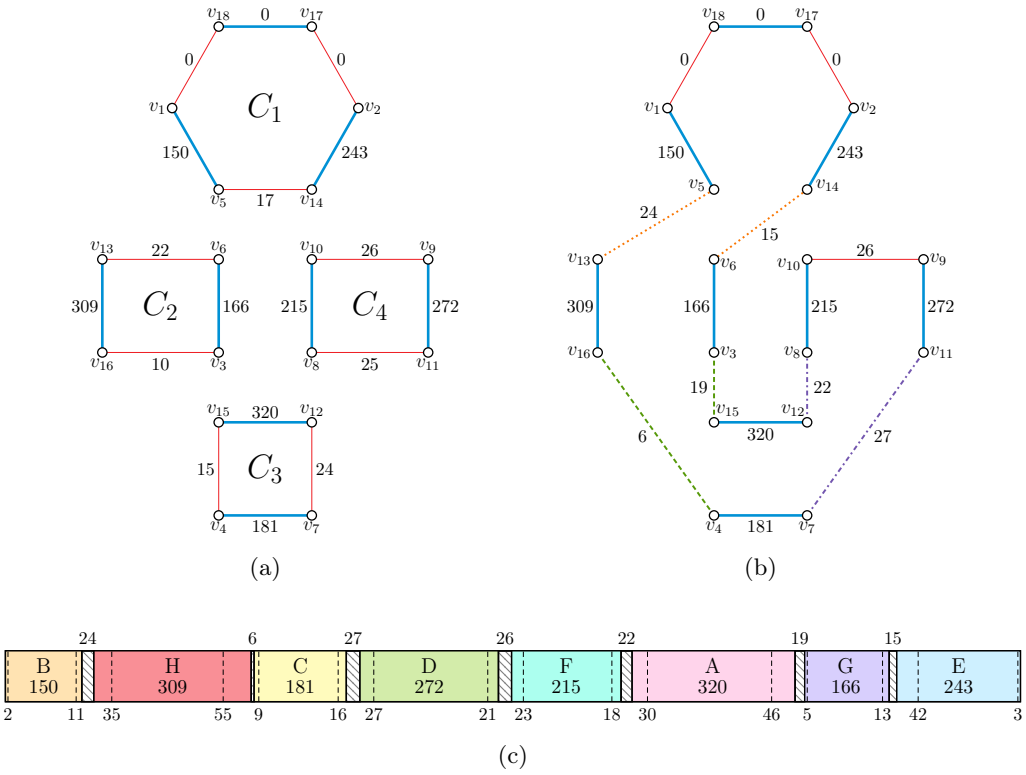


Figure 7.7: (a) The graph $G' = (V, B \cup R')$ comprising four cyclic components; (b) the graph G' using the modified set R' shown in Figure 7.6 which connects the components into a single alternating Hamiltonian cycle; and (c) the corresponding alignment of the eight items in $\mathcal{I}$.

This algorithm for the sub-MSCPP, which will be referred to as AHC′, does not currently have the same guarantee as the original AHC algorithm for the sub-SCPP. Given larger problem instances there may exist a greater number of subsets R''_i; consequently, it may take an extensive period of time to form and assess all possible collections of the subsets. Therefore, at this time we are unable to determine if a minimum length alternating Hamiltonian cycle can be found in polynomial-time for all instances of the sub-MSCPP. This remains an issue warranting further research.

7.3 A Heuristic for the MSCPP

Using the AHC′ algorithm for the sub-MSCPP, we can easily produce a heuristic for the MSCPP based on the MFFD$^+$ heuristic for the SCPP introduced in Chapter 4. This new heuristic, MFFD$^{+\prime}$, operates in a similar manner to MFFD$^+$ as shown in Algorithm 16, where the items are packed in turn into the first bin that can accommodate the item. However, MFFD$^{+\prime}$ must take into account the total inter-item width $f(S)$ of items in a bin S. Therefore, AHC′ is executed to find an arrangement S' of all items in the current bin S_j and the current item i with minimal inter-item width $f(S')$ (Line 9). If the new configuration of items in S' can be packed into a single bin feasibly – that is, the total width of the items in S' and the total inter-item

width $f(S')$ does not exceed the bin capacity W – then MFFD$^{+\prime}$ replaces S_j with the arrangement of items in S' which includes the item i (Lines 10–13). Otherwise, MFFD$^{+\prime}$ attempts to pack i into the next bin S_{j+1}.

Algorithm 16 MFFD$^{+\prime}(\mathcal{I})$

```
1: sort items in I in order of non-increasing widths w_i
2: A(S_j) ← 0 ∀ j = 1, ..., |I|                    ▷ All bins are initially empty
3: S ← ∅
4: for i ← 1 to |I| do
5:     j ← 1
6:     packed ← false
7:     while not packed do
8:         if S_j ∈ S and A(S_j) + w_i ≤ W then
9:             S' ← AHC'(S_j, i)
10:            if A(S') + f(S') ≤ W then
11:                S_j ← S'
12:                A(S_j) ← A(S') + f(S')
13:                packed ← true
14:            else j ← j + 1
15:        else if S_j ∉ S then
16:            S_j ← S_j ∪ (a_i, b_i)              ▷ Item i packed into a new bin
17:            A(S_j) ← A(S_j) + w_i
18:            S ← S ∪ S_j
19:            packed ← true
20: return S
```

7.4 Computational Results

To demonstrate the differences between the SCPP and the MSCPP, we ran the MFFD$^{+\prime}$ heuristic for the MSCPP on our set of problem instances for the SCPP with varying values of δ. Tables 7.1 to 7.3 compare the results obtained by MFFD$^{+\prime}$ for the MSCPP with the results using MFFD^{+} for the SCPP from Chapter 4. The source code and results for the MFFD$^{+\prime}$ heuristic is available online (Hawa, 2020a).

As expected, across all values of δ the MFFD$^{+\prime}$ produces solutions using fewer bins $|\mathcal{S}|$ on average, with notable differences for instances using $W = 5000$ when $\delta = 0.25$. As fewer score widths meet the vicinal sum constraint there are fewer items per bin on average for the SCPP; however, for the MSCPP more items can be packed into a single bin due to the modified vicinal sum constraint, even more so for the larger bin capacity. We also see that when $\delta = 0.25$, MFFD$^{+\prime}$ is able to produce solutions for the MSCPP that comprise t bins for five of the 18 instance classes, whereas MFFD^{+} is unable to produce any such solutions for the SCPP.

For $\delta = 0.5$ and $\delta = 0.75$, although MFFD$^{+\prime}$ does still yield solutions using the fewest bins, the difference between the average number of bins of the two heuristics is smaller, as a higher proportion of score widths meet the vicinal sum constraint.

Table 7.1: Results obtained using the MFFD$^{+\prime}$ and MFFD^{+} heuristics for the MSCPP and the SCPP respectively for $\delta = 0.25$.

$\delta = 0.25$			MFFD$^{+\prime}$		MFFD^{+}	
Type, W	$\lvert\mathcal{I}\rvert$	t^a	$\lvert\mathcal{S}\rvert^b$	$\%t^c$	$\lvert\mathcal{S}\rvert$	$\%t$
a, 1250	100	46.13	48.03 ± 6.1	0.3	59.50 ± 5.4	0.0
	500	229.37	233.733 ± 2.1	0.0	289.84 ± 2.3	0.0
	1000	458.37	465.57 ± 1.4	0.0	574.26 ± 1.7	0.0
a, 2500	100	23.32	23.94 ± 4.4	38.2	46.27 ± 8.6	0.0
	500	114.94	117.74 ± 1.9	0.0	220.87 ± 4.1	0.0
	1000	229.44	234.87 ± 1.3	0.0	435.81 ± 3.1	0.0
a, 5000	100	11.92	12.25 ± 4.7	66.9	42.70 ± 10.4	0.0
	500	57.72	59.37 ± 1.9	0.0	203.56 ± 4.9	0.0
	1000	114.97	118.23 ± 1.3	0.0	402.3 ± 3.8	0.0
r, 1250	100	46.44	52.87 ± 17.7	0.0	65.52 ± 14.1	0.0
	500	229.97	260.15 ± 16.8	0.0	332.46 ± 13.4	0.0
	1000	459.38	519.57 ± 16.7	0.0	646.49 ± 13.4	0.0
r, 2500	100	23.47	24.56 ± 11.3	8.4	51.93 ± 19.6	0.0
	500	115.24	120.95 ± 10.5	0.0	257.99 ± 18.4	0.0
	1000	229.95	241.48 ± 10.5	0.0	516.00 ± 18.4	0.0
r, 5000	100	11.98	12.48 ± 11.2	50.3	49.35 ± 22.3	0.0
	500	57.87	60.53 ± 10.4	0.0	246.96 ± 20.7	0.0
	1000	115.23	120.62 ± 10.4	0.0	494.44 ± 20.7	0.0

[a] $t = \lceil \sum_{i=1}^{n} w_i/W \rceil$ (mean from 1000 instances).
[b] Number of bins per solution (mean from 1000 instances plus or minus the coefficient of variation (%)).
[c] Percentage of instances in which the solution comprises t bins.

Table 7.2: Results obtained using the MFFD$^{+\prime}$ and MFFD^{+} heuristics for the MSCPP and the SCPP respectively for $\delta = 0.5$.

$\delta = 0.5$			MFFD$^{+\prime}$		MFFD^{+}	
Type, W	$\lvert\mathcal{I}\rvert$	t	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$
a, 1250	100	46.13	47.67 ± 6.2	9.2	49.73 ± 5.7	0.8
	500	229.37	232.02 ± 2.1	0.0	239.15 ± 2.3	0.0
	1000	458.37	462.18 ± 1.4	0.0	472.35 ± 1.6	0.0
a, 2500	100	23.32	23.62 ± 4.4	70.1	28.46 ± 10.4	2.6
	500	114.94	115.98 ± 1.9	4.0	132.65 ± 4.9	0.0
	1000	229.44	231.41 ± 1.3	0.0	258.39 ± 3.5	0.0
a, 5000	100	11.92	12.03 ± 4.9	89.7	19.88 ± 18.3	0.7
	500	57.72	58.12 ± 2.0	60.3	89.54 ± 9.3	0.0
	1000	114.97	115.74 ± 1.4	22.1	172.61 ± 7.1	0.0
r, 1250	100	46.44	52.44 ± 17.8	0.0	56.51 ± 15.9	0.0
	500	229.97	257.99 ± 16.9	0.0	278.39 ± 14.9	0.0
	1000	459.38	515.23 ± 16.9	0.0	556.37 ± 14.8	0.0
r, 2500	100	23.47	24.22 ± 11.6	29.5	35.42 ± 23.1	1.6
	500	115.24	119.21 ± 10.8	0.0	177.25 ± 21.2	0.0
	1000	229.95	237.96 ± 10.7	0.0	355.04 ± 21.2	0.0
r, 5000	100	11.98	12.22 ± 11.5	75.7	29.61 ± 32.7	0.5
	500	57.87	59.39 ± 10.6	1.6	153.42 ± 28.9	0.0
	1000	115.23	118.34 ± 10.6	0.0	308.64 ± 28.7	0.0

Table 7.3: Results obtained using the MFFD$^{+\prime}$ and MFFD^{+} heuristics for the MSCPP and the SCPP respectively for $\delta = 0.75$.

$\delta = 0.75$			MFFD$^{+\prime}$		MFFD^{+}	
Type, W	$\lvert\mathcal{I}\rvert$	t	$\lvert\mathcal{S}\rvert$	$\%t$	$\lvert\mathcal{S}\rvert$	$\%t$
a, 1250	100	46.13	47.60 ± 6.2	12.8	47.76 ± 6.1	10.4
	500	229.37	231.81 ± 2.1	0.0	232.04 ± 2.1	0.0
	1000	458.37	461.86 ± 1.4	0.0	462.06 ± 1.4	0.0
a, 2500	100	23.32	23.56 ± 4.5	76.0	23.94 ± 5.5	55.7
	500	114.94	115.68 ± 1.9	26.5	116.09 ± 2.1	21.3
	1000	229.44	230.86 ± 1.3	0.0	231.16 ± 1.4	0.0
a, 5000	100	11.92	11.99 ± 4.9	93.2	12.48 ± 8.7	63.6
	500	57.72	57.97 ± 1.9	75.7	58.56 ± 2.7	50.9
	1000	114.97	115.45 ± 1.4	51.3	115.91 ± 1.6	37.8
r, 1250	100	46.44	52.27 ± 18.0	0.1	53.39 ± 17.2	0.0
	500	229.97	257.20 ± 17.0	0.0	263.03 ± 16.2	0.0
	1000	459.38	513.65 ± 16.9	0.0	525.60 ± 16.1	0.0
r, 2500	100	23.47	24.06 ± 11.7	43.9	27.69 ± 18.4	17.9
	500	115.24	118.43 ± 10.9	0.0	138.56 ± 17.0	0.0
	1000	229.95	236.42 ± 10.9	0.0	277.84 ± 17.1	0.0
r, 5000	100	11.98	12.14 ± 11.5	84.6	18.28 ± 34.8	19.2
	500	57.87	58.83 ± 10.7	15.5	96.73 ± 31.5	1.9
	1000	115.23	117.21 ± 10.7	0.5	195.60 ± 31.4	0.3

Nevertheless, for $\delta = 0.5$ there are four instance classes whereby, unlike MFFD^{+} for the SCPP, MFFD$^{+\prime}$ finds solutions comprising t bins for the MSCPP. A significant case in point is the artificial instance class with $|\mathcal{I}| = 500$ and $W = 5000$, where MFFD$^{+\prime}$ generates over 600 solutions containing t bins. Moreover, although MFFD^{+} does produce solutions using t bins for the SCPP for five instance classes, MFFD$^{+\prime}$ yields a larger number of such solutions – in three of the instance classes the difference between the number of solutions using t bins ranges from 675 to 890.

The same trend is observed when $\delta = 0.75$, where MFFD$^{+\prime}$ yields solutions comprising fewer bins in comparison to MFFD^{+} and also finds a higher number of solutions using t bins. Despite the increase in the quality of solutions produced by MFFD$^{+\prime}$, however, there still exist instance classes in which even MFFD$^{+\prime}$ is unable to generate solutions of t bins. This, of course, could be due to the inaccuracy of t for the MSCPP, or perhaps suggests that a simple heuristic such as MFFD$^{+\prime}$ is not sufficient enough to produce high quality solutions for all instances of the MSCPP.

7.5 Summary

This chapter has introduced the Modified Score-Constrained Packing Problem (MSCPP), an alternative version of the SCPP in which spaces can exists between adjacent items in bins to aid the fulfillment of the vicinal sum constraint. The similarites and dif-

ferences between the MSCPP and the SCPP have been explored, and an algorithm based on the AHC algorithm of Chapter 3 is presented for the sub-MSCPP. From this, we have created the MFFD$^{+\prime}$ heuristic for the MSCPP, which is analogous to MFFD^{+} for the SCPP, and have compared the results of the heuristics to see the differences in solutions of the two problems.

The main advantage of the MSCPP is the increase in the number of items per bin which, of course, can reduce the total number of bins in the final solution. Clearly, this is a desirable factor in industry as fewer bins (i.e. strips of material) will be required, thus reducing costs. However, it still remains to be seen whether all instances of the sub-MSCPP can be solved in polynomial-time.

Chapter 8

Conclusions and Future Research

This thesis has been dedicated to the Score-Constrained Packing Problem (SCPP), a combinatorial optimisation problem that generalises the classical one-dimensional bin packing problem (BPP), in which the goal is to pack a set $\mathcal{I}$ of items that possess score lines into the fewest number of bins such that no bin is overfilled and vicinal sum constraint (1.1) is fulfilled in each bin. As the order and orientation of the items in each bin affects the feasibility of a solution, standard methods for the BPP are not guaranteed to produce valid solutions for the SCPP.

The single bin version of the problem was motivated by a problem arising in the packaging industry, where sheets of cardboard are to be cut and folded into boxes (Goulimis, 2004). Finding high quality solutions to the SCPP not only reduces the cost of materials, but also minimises the amount of waste generated, which should be a priority given current environmental issues. This further motivates the study of the SCPP.

As the SCPP is a relatively new problem, this thesis set out to discover the different properties that benefit the SCPP. To do so, we studied various methods including heuristics, metaheuristics, and exact methods, allowing us to form a better understanding and overview of the problem.

8.1 Summary of Findings

In this section, we present the findings and contributions of this thesis relating to the research aims stated in Section 1.2.

8.1.1 Research Aim 1

To compare several heuristics for the SCPP and identify characteristics that make the heuristics suitable for particular problem instance types.

Due to the novelty of the SCPP and thus the lack of benchmark instances, heuristics are an appropriate initial approach to better understand what is required to produce higher quality solutions. In Chapter 4, we explored three heuristics for the SCPP. The effectiveness and the simplicity of the FFD heuristic for the BPP discussed in Chapter 2 led us to base two of our heuristics on FFD: MFFD, a slightly modified version of FFD, and MFFD$^+$ which incorporates the Alternating Hamiltonian Construction (AHC) algorithm for the sub-SCPP detailed in Chapter 3. We also developed a third heuristic, PS, which focuses on packing individual bins in turn.

A wide range of problem instances for the SCPP with varying item sizes and types as well as bin capacities were used to compare the heuristics. A parameter δ was also introduced to control the proportion of score widths of items in a set $\mathcal{I}$ that satisfy the vicinal sum constraint. This allows us to alter the difficulty of the problem instances and fully assess the performance of the heuristics in a variety of scenarios.

It is immediately evident that the limitations on MFFD proved unfit for the SCPP, producing the lowest quality solutions across the majority of instance classes. The PS heuristic was shown to be beneficial for problem instances where the proportion of score widths that meet the vicinal sum constraint is low, due to the heuristic's mechanism of selecting items based on their score widths. On the other hand, MFFD$^+$ yields higher quality solutions for instances with fewer items per bin and instances with a higher proportion of score widths that meet the vicinal sum constraint. This indicates that MFFD$^+$ is more suited to instances with smaller bin sizes where the widths of the items are important, as PS does not consider the items' widths. The lack of an accurate lower bound for the SCPP is also acknowledged, and suggests that optimal solutions may have been found in a larger number of problem instances. Ultimately, the use of heuristics further confirms that the ability to reorder and reorientate items within bins is crucial in the formation of higher quality solutions.

8.1.2 Research Aim 2

To investigate the effects of different recombination operators within an evolutionary algorithm framework on the quality of solutions and determine the desirable attributes of each operator.

In Chapter 2 we highlighted the abundance of literature of evolutionary algorithms (EAs) for the BPP and related packing problems including the TPP, a problem similar to the SCPP with order and orientation restrictions. The adaptability of EAs is appealing for a problem like the SCPP, because the operators can be tailored

specifically for the constraints of the problem. Chapter 5 describes our EA for the SCPP which includes a local search procedure to introduce the movement of items between bins, an attractive property not seen our the previous heuristics.

Within the EA framework, three distinct recombination operators were compared, each with different priorities. Computational results show that the AGX recombination operator performed worst overall in line with results seen in existing work (Lewis and Holborn, 2017). Although operators that perform in a similar manner to AGX would be suitable for the BPP, the results produced using AGX indicate that the vicinal sum constraint must be prioritised when forming solutions. This is further confirmed by the higher quality results obtained using the AGX$'$ recombination operator which operates by seeking bins containing the most items, and in these scenarios it is difficult to fulfil the vicinal sum constraint; thus such bins are more desirable. The advantage of the AGX$'$ mechanism is particularly noticeable in experiments when $\delta = 0.25$, where the proportion of score widths that meet the vicinal sum constraint is smallest. Furthermore, the GGA recombination operator yields high quality results on average in the majority of instance classes, which suggests that the formation of good groups of bins over a high number of iterations is beneficial for the SCPP.

8.1.3 Research Aim 3

To explore the combination of exact methods with metaheuristics and assess the associated advantages and disadvantages.

Having investigated heuristics and metaheuristics, our next step into helping us better understand the SCPP was to look into exact methods. Chapter 6 provided an ILP formulation for a generalised vehicle routing problem corresponding to the SCPP and discussed how commercial solvers are not appropriate for larger, realistic problem instance sizes. Instead, we described the MXCP, which involves finding an exact cover of minimum cardinality from a collection of subsets. We then introduced two CMSA-based algorithms: CMSA-M and CMSA-EA, which use metaheuristics to form and evolve a set of high quality bins $\mathcal{B}$ that make up high quality solutions for the given instance $\mathcal{I}$. They also use an exact algorithm, MinDLX, to solve the MXCP and find an optimal solution $\mathcal{S}^*$ with respect to $\mathcal{B}$ in each iteration.

The idea behind this approach is that finding a solution will be easier with a smaller, restricted set of bins $\mathcal{B}$. However, it is clear from the outset that, despite the mechanisms and parameters used to control the size of the set of bins $\mathcal{B}$, MinDLX is computationally expensive and the amount of time allocated for the procedure to find an optimal solution with respect to $\mathcal{B}$ is insufficient. This affects the overall

CMSA algorithms, primarily by reducing the number of iterations performed which in turn restricts exploration of the search space, implying that the problem size and complexity of the SCPP is not appropriate for this type of exact method approach. However, in theory the CMSA algorithm framework can be seen to be promising due to its focus on collecting high quality groups of bins, which is a successful approach within our EA in the previous chapter. Moreover, the effectiveness of the feedback mechanism implemented within CMSA-M is highlighted, and further investigation should be considered into exploiting this feature.

8.1.4 Research Aim 4

To determine whether improved solutions can be found by relaxing elements of the SCPP, namely by allowing spaces between items in each bin.

After studying the SCPP, we turned our attention to an alternative version of the problem. Chapter 7 initially defined the Modified Score-Constrained Packing Problem (MSCPP) and its associated sub-problem (sub-MSCPP) which use the modified vicinal sum constraint, a relaxed version of (1.1) that permits spaces between items in a packing so that the distance between score lines of adjacent items is greater than or equal to the minimum scoring distance τ. We then introduced AHC$'$, adapted from the AHC algorithm for the sub-SCPP, which aims to find the shortest alternating Hamiltonian cycle in a graph G modelling an instance of the sub-MSCPP. Using AHC$'$, we created the MFFD$^{+\prime}$ heuristic for the MSCPP, based on MFFD^{+} for the SCPP, and compared the results of these heuristics to see how the solutions for the two problems differ.

We saw that MFFD$^{+\prime}$ produced solutions using the fewest number of bins $|\mathcal{S}|$ on average in all instance classes. Clearly, relaxing the vicinal sum constraint gives rise to an increase in the number of items per bin, resulting in fewer bins in the overall solution. However, it is also important to note that although we are able to deduce a lower bound for the shortest alternating Hamiltonian cycle in G for the sub-MSCPP, at the time of writing there is no guarantee that AHC$'$ can solve all instances of the sub-MSCPP in polynomial-time, as with AHC for the sub-SCPP.

8.2 Future Research

The research presented in this thesis gives rise to a variety of ways in which the study could be extended. In this section, we provide several suggestions for future work considering both theoretical and computational aspects.

8.2.1 Lower Bounds

Throughout this thesis, the theoretical minimum t (2.2) for the BPP is used to analyse results for the SCPP. Although a solution $\mathcal{S}$ for an instance $\mathcal{I}$ of the SCPP is an optimal solution if $|\mathcal{S}| = t$, there is currently no way to determine whether $\mathcal{S}$ is optimal when $|\mathcal{S}| > t$. Of course, other lower bounds that exist in literature could be considered for the SCPP; however, ideally, an accurate lower bound for the SCPP would take into account δ – the proportion of score widths of a set of items $\mathcal{I}$ that fulfil the vicinal sum constraint – as well as the bin capacity W. It has been shown in our experiments that as δ decreases and tends towards 0 the vicinal sum constraint should be prioritised as fewer items can be packed into a single bin, whilst as δ tends to 1 the SCPP becomes equivalent to the BPP; thus the focus must shift to the bin capacity constraint. Having a reliable lower bound for the SCPP would allow us to determine the performance of algorithms in the form of upper bounds and develop approximation algorithms.

8.2.2 Alternative Heuristics

Two of the heuristics examined in this thesis are based upon the FFD heuristic for the BPP, however other heuristics might be considered for the SCPP. There exists an abundance of literature proposing heuristics for the BPP and related packing problems, many of which can be adapted for the SCPP. Investigating a wider variety of methods for the SCPP would provide further insight into the characteristics that improve and hinder the performance of a constructive heuristic. Potential studies range from the basic BFD heuristic (Johnson, 1973) modified to incorporate the AHC algorithm as with MFFD$^+$, to developing guidelines for partitioning items into subsets in a similar manner to the Harmonic$_M$ heuristic (Lee and Lee, 1985).

8.2.3 Evolutionary Algorithm Operators and Metaheuristics

In Chapter 5, where we have developed the first EA for the SCPP, the focus is on comparing the effects of different recombination operators on the quality of solutions. However, there also exists other operators within the EA that should be investigated in a similar manner, including the number of parent solutions chosen from the population in each iteration, how the parents are selected, the number of offspring solutions produced from the parent solutions, alternative distributions for selecting values of r in the mutation operator, as well as considering different local search methods. Moreover, additional heuristics such as MFFD and PS could be used in conjuction with MFFD$^+$ and MFFR$^+$ to create candidate solutions for the initial population. Besides EAs, other metaheuristic methods and properties should

be studied both within and independent of our EA, for example, simulated annealing and tabu search, both of which would provide an invaluable insight into the mechanisms of the algorithms with respect to the SCPP and allow us to examine how efficiently each method navigates the solution space.

8.2.4 Combining Techniques

We saw in Chapter 6 that the exact algorithm MinDLX used within our CMSA-based algorithms requires an unrealistic amount of time in order to perform correctly given the size of the set of bins $\mathcal{B}$. Other than further restricting the size of $\mathcal{B}$, a simple substitute to consider is implementing an ILP solver in place of MinDLX. Alternatively, the exact solver within CMSA could be removed entirely, instead using a procedure to find just a single solution in $\mathcal{B}$ rather than the minimum cardinality exact cover. Finding solutions faster would increase the number of CMSA iterations and encourage the evolution of $\mathcal{B}$. In addition, a method such as tabu search could be implemented to prevent the algorithm from producing identical solutions and bins in each iteration. The feedback mechanism should also be exploited further to introduce new, higher quality bins into $\mathcal{B}$.

8.2.5 The MSCPP

The main focus into the MSCPP should be on the complexity of the AHC$'$ algorithm and/or determining whether there exists an exact polynomial-time algorithm for the sub-MSCPP which would prove beneficial for the MSCPP. Methods for the SCPP could then be used for the MSCPP which would allow us to further examine the similarities and differences between the two problems. Furthermore, an additional constraint of maximising useable leftover materials could be considered, which may have a positive impact in industrial applications.

8.3 Final Remarks

This final chapter has presented a summary of significant findings obtained throughout this thesis and highlights promising areas of our research. The novelty of the SCPP gives rise to numerous avenues for future work regarding both the SCPP and related problems, for which this thesis provides a solid foundation. Further publications of the research in this thesis are also underway.

8.4 Summary of Available Resources

This section provides an overview of the resources created and used for the research in this thesis. All resources are accompanied with a user guide as well as a list of the specific parameter values used for each of the experiments.

- The problem instance generator and all problem instances used in the experiments throughout this thesis can be found at Hawa (2020g).
- The code and results for the three constructive heuristics in Chapter 4 is available at Hawa (2020f).
- The code and results for the evolutionary algorithm framework in Chapter 5, which includes all three recombination operators, is provided by Hawa (2020e).
- The ILP model for the SCPP presented in Chapter 6 is available online (Hawa, 2020c) and is implemented in Xpress Mosel.
- The code and results for the two CMSA-based algorithms described in Chapter 6 is available at Hawa (2020d).
- The ILP model for the sub-MSCPP introduced in Chapter 7 is available online (Hawa, 2020b) and is implemented in Xpress Mosel.
- Finally, the code and results for the MFFD$^{+\prime}$ heuristic for the MSCPP in Chapter 7 can be found at Hawa (2020a).

Bibliography

Aardal, K. I., Van Hoesel, S. P., Koster, A. M., Mannino, C., and Sassano, A. (2007). Models and solution techniques for frequency assignment problems. *Annals of Operations Research*, 153(1):79–129.

Akeb, H., Hifi, M., and M'Hallah, R. (2009). A beam search algorithm for the circular packing problem. *Computers & Operations Research*, 36(5):1513–1528.

Alvarez-Valdes, R., Martinez, A., and Tamarit, J. (2013). A branch & bound algorithm for cutting and packing irregularly shaped pieces. *International Journal of Production Economics*, 145(2):463–477.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (1999). Finding tours in the TSP. Technical report.

Arora, D., Maini, P., Pinacho-Davidson, P., and Blum, C. (2019). Route planning for cooperative air-ground robots with fuel constraints: an approach based on CMSA. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 207–214.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press.

Bäck, T., Fogel, D. B., and Michalewicz, Z. (1997). *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press.

Baker, B. S. (1985). A new proof for the first-fit decreasing bin-packing algorithm. *Journal of Algorithms*, 6(1):49–70.

Baker, B. S., Coffman, Jr, E. G., and Rivest, R. L. (1980). Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855.

Balas, E. (1965). An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13(4):517–546.

Bang-Jensen, J. and Gutin, G. (1997). Alternating cycles and paths in edge-coloured multigraphs: a survey. *Discrete Mathematics*, 165(1):39–60.

Bang-Jensen, J. and Gutin, G. (1998). Alternating cycles and trails in 2-edge-coloured complete multigraphs. *Discrete Mathematics*, 188(1-3):61–72.

Bankfalvi, M. and Bankfalvi, Z. (1968). Alternating Hamiltonian circuit in two-coloured complete graphs. In *Proceedings of Colloq. Tihany 1968*, pages 11–18.

Becker, K. H. (2010). *Twin-Constrained Hamiltonian Paths on Threshold Graphs - An Approach to the Minimum Score Separation Problem.* PhD thesis, London School of Economics.

Bellman, R. E. (1961). Dynamic programming treatment of the traveling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63.

Belov, G. and Scheithauer, G. (2002). A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths. *European Journal of Operational Research*, 141(2):274–294.

Beltrán, J. D., Calderón, J. E., Cabrera, R. J., Moreno-Pérez, J. A., and Moreno-Vega, J. M. (2004). GRASP-VNS hybrid for the Strip Packing Problem. *Hybrid metaheuristics*, 2004:79–90.

Bennell, J. A., Dowsland, K. A., and Dowsland, W. B. (2001). The irregular cutting-stock problem – a new procedure for deriving the no-fit polygon. *Computers & Operations Research*, 28(3):271–287.

Bianchi, L., Birattari, M., Chiarandini, M., Manfrin, M., Mastrolilli, M., Paquete, L., Rossi-Doria, O., and Schiavinotto, T. (2006). Hybrid metaheuristics for the vehicle routing problem with stochastic demands. *Journal of Mathematical Modelling and Algorithms*, 5(1):91–110.

Bianchi, L., Dorigo, M., Gambardella, L. M., and Gutjahr, W. J. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287.

Bjorklund, A. (2014). Determinant sums for undirected Hamiltonicity. *SIAM Journal on Computing*, 43(1):280–299.

Blum, C. (2016). Construct, merge, solve and adapt: application to unbalanced minimum common string partition. In *International Workshop on Hybrid Metaheuristics*, pages 17–31. Springer.

Blum, C. and Blesa, M. J. (2016). Construct, merge, solve and adapt: application to the repetition-free longest common subsequence problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 46–57. Springer.

Blum, C. and Calvo, B. (2015). A matheuristic for the minimum weight rooted arborescence problem. *Journal of Heuristics*, 21(4):479–499.

Blum, C. and Pereira, J. (2016). Extension of the CMSA algorithm: an LP-based way for reducing sub-instances. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 285–292.

Blum, C., Pinacho, P., López-Ibáñez, M., and Lozano, J. A. (2016). Construct, merge, solve & adapt a new general algorithm for combinatorial optimization. *Computers & Operations Research*, 68:75–88.

Blum, C., Puchinger, J., Raidl, G. R., and Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: a survey. *Applied Soft Computing*, 11(6):4135–4151.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308.

Blum, C., Roli, A., and Sampels, M. (2008). *Hybrid metaheuristics: an emerging approach to optimization*, volume 114. Springer.

Blum, C. and Santos, H. G. (2019). Generic CP-supported CMSA for binary integer linear programs. In *International Workshop on Hybrid Metaheuristics*, pages 1–15. Springer.

Blum, C. and Schmid, V. (2013). Solving the 2D bin packing problem by means of a hybrid evolutionary algorithm. *Procedia Computer Science*, 18:899–908.

Bollobás, B. and Erdös, P. (1976). Alternating Hamiltonian cycles. *Israel Journal of Mathematics*, 23(2):126–131.

Bortfeldt, A. (2006). A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research*, 172(3):814–837.

Brandstadt, A., Spinrad, J. P., et al. (1999). *Graph classes: a survey*, volume 3. SIAM.

Brown, A. R. (1971). *Optimum packing and depletion*. American Elsevier, New York.

Buckles, B. P. and Petry, F. E. (1992). *Genetic Algorithms*. IEEE Computer Society Press.

Burke, E. K., Hyde, M. R., and Kendall, G. (2006). Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature-PPSN IX*, pages 860–869. Springer.

Cardoso Silva, A. and Hasenclever Borges, C. C. (2019). An improved heuristic based genetic algorithm for bin packing problem. In *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 60–65. IEEE.

Carter, M. W., Laporte, G., and Lee, S. Y. (1996). Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 47(3):373–383.

Chan, L. M. A., Simchi-Levi, D., and Bramel, J. (1998). Worst-case analyses, linear programming and the bin-packing problem. *Mathematical Programming*, 83(1-3):213–227.

Chandra, A., Hirschberg, D. S., and Wong, C. (1978). Bin packing with geometric constraints in computer network design. *Operations Research*, 26(5):760–772.

Chen, C. and Daykin, D. E. (1976). Graphs with Hamiltonian cycles having adjacent lines different colors. *Journal of Combinatorial Theory, Series B*, 21(2):135–139.

Chen, C.-L. and Chen, C.-L. (2009). Hybrid metaheuristics for unrelated parallel machine scheduling with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 43(1-2):161.

Chvátal, V. and Hammer, P. L. (1973). Set packing and threshold graphs. *Waterloo Res. Report*, pages 73–21.

Chvátal, V. and Hammer, P. L. (1977). Aggregation of inequalities in integer programming. *Annals of Discrete Mathematics*, 1:145–162.

Cobham, A. (1965). The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland Publishing Company: Amsterdam.

Coelho, K. R., Cherri, A. C., Baptista, E. C., Jabbour, C. J. C., and Soler, E. M. (2017). Sustainable operations: The cutting stock problem with usable leftovers from a sustainable perspective. *Journal of cleaner production*, 167:545–552.

Coffman, E. G., Csirik, J., Galambos, G., Martello, S., and Vigo, D. (2013). Bin packing approximation algorithms: survey and classification. *Handbook of combinatorial optimization*, pages 455–531.

Coffman, E. G., Galambos, G., Martello, S., and Vigo, D. (1999). Bin packing approximation algorithms: Combinatorial analysis. In *Handbook of combinatorial optimization*, pages 151–207. Springer.

Coffman, E. G., Garey, M. R., and Johnson, D. S. (1978). An Application of Bin-Packing to Multiprocessor Scheduling. *SIAM Journal on Computing*, 7(1):1–17.

Coffman, E. G., Garey, M. R., and Johnson, D. S. (1984). Approximation Algorithms for Bin-Packing – An Updated Survey. In *Algorithm Design for Computer System Design*, pages 49–106. Springer.

Coffman, E. G., Garey, M. R., and Johnson, D. S. (1987). Bin packing with divisible item sizes. *Journal of Complexity*, 3(4):406–428.

Coffman, E. G., Garey, M. R., and Johnson, D. S. (1996). Approximation algorithms for bin packing: a survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co.

Cook, S. (2006). The P versus NP problem. *The millennium prize problems*, pages 87–104.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of Computing*, pages 151–158.

Cook, S. A. (1983). An overview of computational complexity. *Communications of the ACM*, 26(6):400–408.

Cook, W. and Seymour, P. (2003). Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248.

Coudert, D., Nepomuceno, N., and Rivano, H. (2010). Power-efficient radio configuration in fixed broadband wireless networks. *Computer Communications*, 33(8):898–906.

Coudert, D., Nepomuceno, N., and Tahiri, I. (2011). Energy saving in fixed wireless broadband networks. In *International Conference on Network Optimization*, pages 484–489. Springer.

Csirik, J. and Totik, V. (1988). Online algorithms for a dual version of bin packing. *Discrete Applied Mathematics*, 21(2):163–167.

Cui, Y., Song, X., Chen, Y., and Cui, Y.-P. (2017). New model and heuristic solution approach for one-dimensional cutting stock problem with usable leftovers. *Journal of the Operational Research Society*, 68(3):269–280.

Cui, Y., Yang, L., and Chen, Q. (2013). Heuristic for the rectangular strip packing problem with rotation of items. *Computers & Operations Research*, 40(4):1094–1099.

da Silveira, J. L., Miyazawa, F. K., and Xavier, E. C. (2013). Heuristics for the strip packing problem with unloading constraints. *Computers & Operations Research*, 40(4):991–1003.

Darwin, C. (1875). *The variation of animals and plants under domestication.* John Murray.

Daykin, D. (1976). Graphs with cycles having adjacent lines different colors. *Journal of Combinatorial Theory, Series B*, 20(2):149–152.

Delahaye, J.-P. (2006). The science behind Sudoku. *Scientific American*, 294(6):80–87.

Delorme, M., Iori, M., and Martello, S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20.

Delorme, X., Gandibleux, X., and Degoutin, F. (2010). Evolutionary, constructive and hybrid procedures for the bi-objective set packing problem. *European Journal of Operational Research*, 204(2):206–217.

do Nascimento, D., de Araujo, S., and Cherri, A. (2020). Integrated lot-sizing and one-dimensional cutting stock problem with usable leftovers. *Annals of Operations Research*, pages 1–19.

Dorigo, M. (1992). *Optimization, learning and natural algorithms.* PhD thesis, Politecnico di Milano.

Dósa, G. (2007). The Tight Bound of First Fit Decreasing Bin-Packing Algorithm is $FFD(I) \leq 11/9OPT(I) + 6/9$. *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11.

Dósa, G., Li, R., Han, X., and Tuza, Z. (2013). Tight absolute bound for First Fit Decreasing bin-packing: $FFD(l) \leq 11/9OPT(L) + 6/9$. *Theoretical Computer Science*, 510:13–61.

Dósa, G. and Sgall, J. (2013). First Fit bin packing: A tight analysis. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Dósa, G. and Sgall, J. (2014). Optimal analysis of Best Fit bin packing. In *International Colloquium on Automata, Languages, and Programming*, pages 429–441. Springer.

Ecker, K. and Zaks, S. (1977). *On a graph labelling problem.* Ges. f. Math. u. Datenverarb.

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467.

Eilon, S. and Christofides, N. (1971). The Loading Problem. *Management Science*, 17(5):259–268.

Falkenauer, E. (1993). The grouping genetic algorithms: widening the scope of the GA's. *JORBEL-Belgian Journal of Operations Research, Statistics, and Computer Science*, 33(1-2):79–102.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(1):5–30.

Falkenauer, E. (1998). *Genetic algorithms and grouping problems.* John Wiley & Sons, Inc.

Falkenauer, E. and Bouffouix, S. (1991). A genetic algorithm for job shop. In *ICRA*, pages 824–829. Citeseer.

Falkenauer, E. and Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing. In *IEEE International Conference on Robotics and Automation*, pages 1186–1192. IEEE.

Fan, Y., Chu, J., and Xu, H. (2020). Improvement grouping genetic algorithm for solving the bin packing problem. In *Journal of Physics: Conference Series*, volume 1550, page 032168. IOP Publishing.

Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133.

Fernandez de la Vega, W. and Lueker, G. S. (1981). Bin packing can be solved within $1+\varepsilon$ in linear time. *Combinatorica*, 1(4):349–355.

Ferreira, C. (2001). Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129.

Fidanova, S. and Pop, P. (2016). An improved hybrid ant-local search algorithm for the partition graph coloring problem. *Journal of Computational and Applied Mathematics*, 293:55–61.

Fleszar, K. and Hindi, K. S. (2002). New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29(7):821–839.

Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial intelligence through simulated evolution*. John Wiley & Sons.

Galinier, P. and Hao, J.-K. (1999). Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397.

Garey, M. R., Graham, R. L., Johnson, D. S., and Yao, A. C.-C. (1976). Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257–298.

Garey, M. R., Graham, R. L., and Ullman, J. D. (1972). Worst-case analysis of memory allocation algorithms. In *Proceedings of the fourth annual ACM Symposium on Theory of Computing*, pages 143–150. ACM.

Garey, M. R. and Johnson, D. S. (1978). "Strong"NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM*, 25(3):499–508.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman Co., San Francisco.

Garraffa, M., Salassa, F., Vancroonenburg, W., Vanden Berghe, G., and Wauters, T. (2016). The one-dimensional cutting stock problem with sequence-dependent cut losses. *International Transactions in Operational Research*, 23(1-2):5–24.

Gilmore, P. C. and Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859.

Gilmore, P. C. and Gomory, R. E. (1963). A linear programming approach to the cutting stock problem—Part II. *Operations Research*, 11(6):863–888.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549.

Glover, F. W. and Kochenberger, G. A. (2003). *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Co.

Golumbic, M. C. (2004). *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier.

Gould, R. J. (1991). Updating the Hamiltonian problem – a survey. *Journal of Graph Theory*, 15(2):121–157.

Gould, R. J. (2003). Advances on the Hamiltonian problem – a survey. *Graphs and Combinatorics*, 19(1):7–52.

Goulimis, C. (2004). Minimum Score Separation - an open combinatorial problem associated with the cutting stock problem. *Journal of the Operational Research Society*, 55(12):1367–1368.

Grossman, J. W. and Häggkvist, R. (1983). Alternating cycles in edge-partitioned graphs. *Journal of Combinatorial Theory, Series B*, 34(1):77–81.

Gupta, J. N. and Ho, J. C. (1999). A new heuristic algorithm for the one-dimensional bin-packing problem. *Production Planning & Control*, 10(6):598–603.

Häggkvist, R. (1977). *On F-Hamiltonian graphs*. University of Umeå, Department of Mathematics.

Haouari, M. and Serairi, M. (2009). Heuristics for the variable sized bin-packing problem. *Computers & Operations Research*, 36(10):2877–2884.

Hawa, A. L. (2020a). A Heuristic for the Modified Score-Constrained Packing Problem (MSCPP): Source Code and Results. `https://doi.org/10.5281/zenodo.3986642`.

Hawa, A. L. (2020b). An Integer Linear Programming (ILP) Model for the Modified Score-Constrained Packing Sub-Problem (sub-MSCPP). `https://doi.org/10.5281/zenodo.3986656`.

Hawa, A. L. (2020c). An Integer Linear Programming (ILP) Model for the Score-Constrained Packing Problem (SCPP). `https://doi.org/10.5281/zenodo.3986644`.

Hawa, A. L. (2020d). CMSA Algorithms for the Score-Constrained Packing Problem (SCPP): Source Code and Results. `https://doi.org/10.5281/zenodo.3986637`.

Hawa, A. L. (2020e). Evolutionary Algorithm (EA) for the Score-Constrained Packing Problem (SCPP): Source Code and Results. `https://doi.org/10.5281/zenodo.3986640`.

Hawa, A. L. (2020f). Heuristics for the Score-Constrained Packing Problem (SCPP): Source Code and Results. `https://doi.org/10.5281/zenodo.3986638`.

Hawa, A. L. (2020g). Problem instance generator for the Score-Constrained Packing Problem (SCPP. https://doi.org/10.5281/zenodo.3986636.

Hawa, A. L., Lewis, R., and Thompson, J. M. (2018). Heuristics for the Score-Constrained Strip-Packing Problem. In *International Conference on Combinatorial Optimization and Applications*, pages 449–462. Springer.

He, K., Jin, Y., and Huang, W. (2013). Heuristics for two-dimensional strip packing problem with 90° rotations. *Expert Systems with Applications*, 40(14):5542–5550.

Held, M. and Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, 10(1):196–210.

Hemmelmayr, V., Schmid, V., and Blum, C. (2012). Variable neighbourhood search for the variable sized bin packing problem. *Computers & Operations Research*, 39(5):1097–1108.

Henderson, P. B. and Zalcstein, Y. (1977). A graph-theoretic characterization of the PV_{chunk} class of synchronizing primitives. *SIAM Journal on Computing*, 6(1):88–108.

Hifi, M. (1997). A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems. *Journal of the Operational Research Society*, 48(6):612–622.

Hifi, M. (1998). Exact algorithms for the guillotine strip cutting/packing problem. *Computers & Operations Research*, 25(11):925–940.

Hilton, A. J. (1992). Alternating Hamiltonian circuits in edge-coloured bipartite graphs. *Discrete Applied Mathematics*, 35(3):271–273.

Hinterding, R. and Khan, L. (1994). Genetic algorithms for cutting stock problems: with and without contiguity. In *Progress in evolutionary computation*, pages 166–186. Springer.

Holland, J. H. (1962a). Concerning efficient adaptive systems. In *Self-Organizing Systems*, volume 230, pages 215–230. Spartan Books.

Holland, J. H. (1962b). Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, 9(3):297–314.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press.

Hung, M. S. and Brown, J. R. (1978). An algorithm for a class of loading problems. *Naval Research Logistics (NRL)*, 25(2):289–297.

Iima, H. and Yakawa, T. (2003). A new design of genetic algorithm for bin packing. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, volume 2, pages 1044–1049. IEEE.

Jansen, K. and van Stee, R. (2005). On strip packing with rotations. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 755–761.

Johnson, D. S. (1973). *Near-optimal bin packing algorithms.* PhD thesis, Massachusetts Institute of Technology.

Johnson, D. S. (1974). Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272–314.

Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., and Graham, R. L. (1974). Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. *SIAM Journal on Computing*, 3(4):299–325.

Jongen, H. T., Meer, K., and Triesch, E. (2007). *Optimization theory.* Springer Science & Business Media.

Junkermeier, J. (2015). A genetic algorithm for the bin packing problem. *Evolutionary Computation, St. Cloud State University, Spring.*

Kammarti, R., Ayachi, I., Ksouri, M., and Borne, P. (2013). Evolutionary approach for the containers bin-packing problem. *Studies in Informatics and Control*, 18(4):315–324.

Kantorovich, L. V. (1960). Mathematical methods of organizing and planning production. *Management science, English translation of a 1939 paper written in Russian*, 6(4):366–422.

Karp, R. M. (1972). Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. Springer.

Kenmochi, M., Imamichi, T., Nonobe, K., Yagiura, M., and Nagamochi, H. (2009). Exact algorithms for the two-dimensional strip packing problem with and without rotations. *European Journal of Operational Research*, 198(1):73–83.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE.

Kierkosz, I. and Luczak, M. (2014). A hybrid evolutionary algorithm for the two-dimensional packing problem. *Central European Journal of Operations Research*, 22(4):729–753.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Klau, G. W., Ljubić, I., Moser, A., Mutzel, P., Neuner, P., Pferschy, U., Raidl, G., and Weiskircher, R. (2004). Combining a memetic algorithm with integer programming to solve the prize-collecting steiner tree problem. In *Genetic and Evolutionary Computation Conference*, pages 1304–1315. Springer.

Knuth, D. E. (2000). Dancing links. *arXiv preprint cs/0011047*.

Koren, M. (1973). Extreme degree sequences of simple graphs. *Journal of Combinatorial Theory, Series B*, 15(3):213–224.

Korf, R. E. (2002). A New Algorithm for Optimal Bin Packing. In *AAAI/IAAI*, pages 731–736.

Korf, R. E. (2003). An improved algorithm for optimal bin packing. In *IJCAI*, volume 3, pages 1252–1258.

Kröger, B. (1995). Guillotineable bin packing: A genetic approach. *European Journal of Operational Research*, 84(3):645–661.

Kucukyilmaz, T. and Kiziloz, H. E. (2018). Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem. *Computers & Industrial Engineering*, 125:157–170.

Lee, C. C. and Lee, D.-T. (1985). A simple on-line bin-packing algorithm. *Journal of the ACM*, 32(3):562–572.

Leung, S. C., Zhang, D., Zhou, C., and Wu, T. (2012). A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem. *Computers & Operations Research*, 39(1):64–73.

Levine, J. and Ducatelle, F. (2004). Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55(7):705–716.

Lewis, R. (2009). A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Computers & Operations Research*, 36(7):2295–2310.

Lewis, R. (2015a). Graph coloring and recombination. In *Springer Handbook of Computational Intelligence*, pages 1239–1254. Springer.

Lewis, R. (2015b). *A guide to graph colouring*, volume 7. Springer.

Lewis, R. and Holborn, P. (2017). How to Pack Trapezoids: Exact and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 21(3):463–476.

Lewis, R. and Paechter, B. (2007). Finding feasible timetables using group-based operators. *IEEE Transactions on Evolutionary Computation*, 11(3):397–413.

Lewis, R., Song, X., Dowsland, K., and Thompson, J. (2011). An investigation into two bin packing problems with ordering and orientation implications. *European Journal of Operational Research*, 213(1):52–65.

Lewis, R., Thiruvady, D., and Morgan, K. (2019). Finding happiness: An analysis of the maximum happy vertices problem. *Computers & Operations Research*, 103:265–276.

Lewis, R., Thompson, J., Mumford, C., and Gillard, J. (2012). A wide-ranging computational comparison of high-performance graph colouring algorithms. *Computers & Operations Research*, 39(9):1933–1950.

Lin, S.-W., Lee, Z.-J., Ying, K.-C., and Lee, C.-Y. (2009). Applying hybrid metaheuristics for capacitated vehicle routing problem. *Expert Systems with Applications*, 36(2):1505–1512.

Lizárraga, E., Blesa, M. J., and Blum, C. (2017). Construct, merge, solve and adapt versus large neighborhood search for solving the multi-dimensional knapsack problem: Which one works better when? In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 60–74. Springer.

Lodi, A., Martello, S., and Monaci, M. (2002). Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252.

Lodi, A., Martello, S., and Vigo, D. (1999). Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112(1):158–166.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). Iterated local search. In *Handbook of Metaheuristics*, pages 320–353. Kluwer Academic Publishers.

Lü, Z. and Hao, J.-K. (2010). A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241–250.

Ma, N., Liu, Y., and Zhou, Z. (2019). Two heuristics for the capacitated multi-period cutting stock problem with pattern setup cost. *Computers & Operations Research*, 109:218–229.

Mahadev, N. V. and Peled, U. N. (1994). Longest cycles in threshold graphs. *Discrete Mathematics*, 135(1-3):169–176.

Mahadev, N. V. R. and Peled, U. N. (1995). *Threshold Graphs and Related Topics, Annals of Discrete Mathematics*, volume 56. Elsevier.

Malaguti, E., Monaci, M., and Toth, P. (2008). A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302–316.

Maniezzo, V., Stützle, T., and Voß, S. (2010). *Matheuristics*. Springer.

Martello, S. (1983). An enumerative algorithm for finding Hamiltonian circuits in a directed graph. *ACM Transactions on Mathematical Software (TOMS)*, 9(1):131–138.

Martello, S., Pisinger, D., and Vigo, D. (2000). The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267.

Martello, S. and Toth, P. (1990a). *Knapsack problems: Algorithms and computer implementations*. Wiley.

Martello, S. and Toth, P. (1990b). Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28(1):59–70.

Massen, F., Deville, Y., and Van Hentenryck, P. (2012). Pheromone-based heuristic column generation for vehicle routing problems with black box feasibility. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 260–274. Springer.

Massen, F., López-Ibánez, M., Stützle, T., and Deville, Y. (2013). Experimental analysis of pheromone-based heuristic column generation using irace. In *International Workshop on Hybrid Metaheuristics*, pages 92–106. Springer.

Mathews, G. B. (1896). On the partition of numbers. *Proceedings of the London Mathematical Society*, 1(1):486–490.

Mitchell, M. (1996). *An introduction to genetic algorithms*. MIT press.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Moalic, L. and Gondran, A. (2018). Variations on memetic algorithms for graph coloring problems. *Journal of Heuristics*, 24(1):1–24.

Moscato, P. et al. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989.

Nepomuceno, N., Pinheiro, P., and Coelho, A. L. (2007a). Tackling the container loading problem: a hybrid approach based on integer linear programming and genetic algorithms. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 154–165. Springer.

Nepomuceno, N., Pinheiro, P., and Coelho, A. L. (2008). A hybrid optimization framework for cutting and packing problems. In *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, pages 87–99. Springer.

Nepomuceno, N. V., Pinheiro, P. R., and Coelho, A. L. (2007b). Combining metaheuristics and integer linear programming: a hybrid methodology applied to the container loading problem. In *Proceedings of the XX congreso da sociedade brasileira de computação, concurso de teses e dissertações*, pages 2028–32.

Nouri, H. E., Driss, O. B., and Ghédira, K. (2016). Hybrid metaheuristics for scheduling of machines and transport robots in job shop environment. *Applied Intelligence*, 45(3):808–828.

Ntene, N. and van Vuuren, J. H. (2008). A survey and comparison of heuristics for the 2D oriented on-line strip packing problem. *ORiON*, 24(2):157–183.

Orlin, J. (1977). The minimal integral separator of a threshold graph. In *Annals of Discrete Mathematics*, volume 1, pages 415–419. Elsevier.

Ozcan, S. O., Dokeroglu, T., Cosar, A., and Yazici, A. (2016). A novel grouping genetic algorithm for the one-dimensional bin packing problem on gpu. In *International Symposium on Computer and Information Sciences*, pages 52–60. Springer.

Pankratz, G. (2005). Dynamic vehicle routing by means of a genetic algorithm. *International Journal of Physical Distribution & Logistics Management*.

Papadimitriou, C. H. (2003). *Computational complexity*. John Wiley & Sons Ltd.

Petzold, C. (2008). *The Annotated Turing: a guided tour through Alan Turing's historic paper on computability and the Turing machine*. Wiley.

Pinacho-Davidson, P., Blum, C., and Lozano, J. A. (2018). The weighted independent domination problem: Integer linear programming models and metaheuristic approaches. *European Journal of Operational Research*, 265(3):860–871.

Pinacho-Davidson, P., Bouamama, S., and Blum, C. (2019). Application of CMSA to the minimum capacitated dominating set problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 321–328.

Pinheiro, P. R., Coelho, A. L., de Aguiar, A. B., and Bonates, T. O. (2011). On the concept of density control and its application to a hybrid optimization framework: investigation into cutting problems. *Computers & Industrial Engineering*, 61(3):463–472.

Pinheiro, P. R., Coelho, A. L. V., Aguiar, A. B., and Sobreira Neto, A. d. M. (2012). Towards aid by generate and solve methodology: application in the problem of coverage and connectivity in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 8(10):790459.

Poldi, K. C. and Arenales, M. N. (2009). Heuristics for the one-dimensional cutting stock problem with limited multiple stock lengths. *Computers & Operations Research*, 36(6):2074–2081.

Qin, J., Yin, Y., and Ban, X. (2011). Hybrid discrete particle swarm algorithm for graph coloring problem. *JCP*, 6(6):1175–1182.

Quiroz-Castellanos, M., Cruz-Reyes, L., Torres-Jimenez, J., Gómez, C., Huacuja, H. J. F., and Alvim, A. C. (2015). A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Computers & Operations Research*, 55:52–64.

Radcliffe, N. J. et al. (1991). Forma analysis and random respectful recombination. In *ICGA*, volume 91, pages 222–229.

Reeves, C. (1996). Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, 63(3):371–396.

Rekiek, B., De Lit, P., Pellichero, F., Falkenauer, E., and Delchambre, A. (1999). Applying the equal piles problem to balance assembly lines. In *Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning (ISATP'99)*, pages 399–404. IEEE.

Rivin, I., Vardi, I., and Zimmermann, P. (1994). The n-queens problem. *The American Mathematical Monthly*, 101(7):629–639.

Rohlfshagen, P. and Bullinaria, J. A. (2007). A genetic algorithm with exon shuffling crossover for hard bin packing problems. In *Proceedings of the 9th annual conference on genetic and evolutionary computation*, pages 1365–1371. ACM.

Rubin, F. (1974). A search procedure for Hamilton paths and circuits. *Journal of the ACM (JACM)*, 21(4):576–580.

Saraiva, R. D., Nepomuceno, N., and Rogério Pinheiro, P. (2019). A two-phase approach for single container loading with weakly heterogeneous boxes. *Algorithms*, 12(4):67.

Saraiva, R. D., Nepomuceno, N. V., and Pinheiro, P. R. (2013). The generate-and-solve framework revisited: generating by simulated annealing. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 262–273. Springer.

Scholl, A., Klein, R., and Jürgens, C. (1997). Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627–645.

Schreiber, E. L. and Korf, R. E. (2013). Improved bin completion for optimal bin packing and number partitioning. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc.

Schwerin, P. and Wäscher, G. (1999). A new lower bound for the bin-packing problem and its integration into MTP. *Pesquisa Operacional*, 19(2):111–129.

Scott, D. (1958). *Programming a Combinatorial Puzzle*. Princeton University Department of Electrical Engineering.

Seiden, S. S. (2002). On the online bin packing problem. *Journal of the ACM*, 49(5):640–671.

Singh, A. and Gupta, A. K. (2007). Two heuristics for the one-dimensional bin-packing problem. *OR Spectrum*, 29(4):765–781.

Smith, S. F. (1980). A learning system based on genetic adaptive algorithms.

Song, X. and Bennell, J. A. (2014). Column generation and sequential heuristic procedure for solving an irregular shape cutting stock problem. *Journal of the Operational Research Society*, 65(7):1037–1052.

Sörensen, K. and Glover, F. (2013). Metaheuristics. *Encyclopedia of Operations Research and Management Science*, 62:960–970.

Sörensen, K., Sevaux, M., and Glover, F. (2017). A history of metaheuristics. *arXiv preprint arXiv:1704.00853*.

Storn, R. and Price, K. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann Publishers.

Talbi, E.-G. (2002). A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5):541–564.

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation.* John Wiley & Sons.

Talbi, E.-G. (2013). *Hybrid metaheuristics, Studies in computational intelligence.* Springer-Verlag.

Tarantilis, C. D., Zachariadis, E. E., and Kiranoudis, C. T. (2009). A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. *IEEE Transactions on Intelligent Transportation Systems*, 10(2):255–271.

Thiruvady, D., Blum, C., and Ernst, A. T. (2019). Maximising the net present value of project schedules using CMSA and parallel ACO. In *International Workshop on Hybrid Metaheuristics*, pages 16–30. Springer.

Thiruvady, D. R., Meyer, B., and Ernst, A. T. (2008). Strip packing with hybrid ACO: Placement order is learnable. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1207–1213. IEEE.

Thompson, J. M. and Dowsland, K. A. (1998). A robust simulated annealing based examination timetabling system. *Computers & Operations Research*, 25(7-8):637–648.

Ullman, J. D. (1971). The performance of a memory allocation algorithm. *Princeton University, Department of Electrical Engineering.*

van de Vel, H. and Shijie, S. (1991). An application of the bin-packing technique to job scheduling on uniform processors. *Journal of the Operational Research Society*, 42(2):169–172.

Vazirani, V. V. (2003). *Approximation Algorithms*. Springer-Verlag Berlin Heidelberg.

Wäscher, G., Haußner, H., and Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130.

Williamson, D. P. and Shmoys, D. B. (2011). *The design of approximation algorithms*. Cambridge University Press.

Wu, D. and Yan, C. (2016). A balance approach for the one-dimensional multiple stock size cutting stock problem with setup cost. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 230(12):2182–2189.

Xavier, E. and Miyazawa, F. K. (2008). A one-dimensional bin packing problem with shelf divisions. *Discrete Applied Mathematics*, 156(7):1083–1096.

Xia, B. and Tan, Z. (2010). Tighter bounds of the First Fit algorithm for the bin-packing problem. *Discrete Applied Mathematics*, 158(15):1668–1675.

Yang, Z. (1999). On F-Hamiltonian graphs. *Discrete Mathematics*, 196(1-3):281–286.

Yao, A. C.-C. (1980). New algorithms for bin packing. *Journal of the ACM (JACM)*, 27(2):207–227.

Yu, B., Yang, Z., and Yao, B. (2011). A hybrid algorithm for vehicle routing problem with time windows. *Expert Systems with Applications*, 38(1):435–441.

Yue, M. (1991). A simple proof of the inequality $FFD(L) \leq 11/9OPT(L)+1, \forall L$ for the FFD bin-packing algorithm. *Acta Mathematicae Applicatae Sinica*, 7(4):321–331.

Zhang, D., Che, Y., Ye, F., Si, Y.-W., and Leung, S. C. (2016). A hybrid algorithm based on variable neighbourhood for the strip packing problem. *Journal of Combinatorial Optimization*, 32(2):513–530.

Zhou, C., Wu, C., and Feng, Y. (2009). An exact algorithm for the type-constrained and variable sized bin packing problem. *Annals of Operations Research*, 172(1):193.