Exact and Approximate Methods for the Score-Constrained Packing Problem

Asyl L. Hawa[a,*], Rhyd Lewis[b], Jonathan M. Thompson[b]

[a] *Southampton Business School, University of Southampton, Southampton SO17 1BJ, UK*
[b] *School of Mathematics, Cardiff University, Senghennydd Road, Cardiff, CF24 4AG, UK*

Abstract

This paper investigates a packing problem related to the one-dimensional bin packing problem in which the order and orientation of items influences the feasibility of a solution. We give an exact polynomial-time algorithm for the Constrained Ordering Problem, explaining how it can be used to find a feasible packing of items in a single bin. We then introduce an evolutionary algorithm for the multi-bin version of the problem, which incorporates the exact algorithm along with a local search procedure and three recombination operators. The mechanisms and results produced by each of the recombination operators are compared, and we discuss the circumstances in which each approach proves most advantageous.

Keywords: Evolutionary computations, packing, combinatorial optimization

*Corresponding author
Email addresses: A.Hawa@soton.ac.uk (Asyl L. Hawa), LewisR9@cardiff.ac.uk (Rhyd Lewis), ThompsonJM1@cardiff.ac.uk (Jonathan M. Thompson)

1. Introduction

Many problems in operational research and discrete mathematics involve the grouping of elements into subsets. These types of problems can be seen in areas such as scheduling (Thompson & Dowsland, 1998; Carter et al., 1996), frequency assignment (Aardal et al., 2007), graph colouring (Lewis, 2015; 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 in such problems is to produce a partition $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ such that

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

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

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

Here, Constraints (1a) and (1b) state the requirement that every element must be in exactly one of the k subsets, whilst (1c) specifies that each subset $S_j \in \mathcal{S}$ must be feasible, where $\mathcal{F}$ denotes the set of all feasible subsets of elements. The notion of feasibility is dependent on the particular constraints of the given problem. For example, in the graph colouring problem where vertices of a graph must be assigned colours such that no two adjacent vertices are in the same colour class, $\mathcal{F}$ contains all possible independent sets of vertices, whilst for the classical one-dimensional bin-packing problem (BPP) which requires a set of items of varying sizes to be packed into the fewest number of bins of fixed capacity, a bin S_j is feasible only if the sum of its item sizes is less than or equal to the bin's capacity.

The focus of this paper is on a special type of packing problem that occurs in the packaging industry, where flat rectangular items of varying widths are to be cut and scored from fixed-length strips of cardboard which are then folded into boxes.

Consider a set $\mathcal{I}$ of n rectangular items of fixed height H. Each item $i \in \mathcal{I}$ has width $w_i \in \mathbb{Z}^+$, and is marked with two vertical score lines in predetermined places. The distances between each score line and the nearest edge of the item are known as 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 Fig. 1. In this industrial process, pairs of knives mounted on a bar simultaneously cut along the score lines of two adjacent items, making it easier to fold the cardboard at a later stage; however, due to the manner in which the machine is designed, the knives in each pair must maintain a minimum distance from one another – a so-called "minimum scoring distance" $\tau \in \mathbb{Z}^+$ (approximately 70mm in practice). For the knives to score all of the items in the correct locations, the distance between two score lines of adjacent items must therefore equal or exceed the minimum scoring distance. Hence, the following *vicinal sum constraint* must be fulfilled:

$$\mathbf{rhs}(i) + \mathbf{lhs}(i+1) \geq \tau \quad \forall\, i \in \{1, 2, \ldots, |S| - 1\}, \tag{2}$$

where $\mathbf{lhs}(i)$ and $\mathbf{rhs}(i)$ denote the left- and right-hand score widths of the ith item in bin S. Clearly, if this constraint is satisfied, the distance between the score lines will be

sufficient for the knives to be able to cut appropriately. An example of this is also shown in Fig. 1. Here, although the vicinal sum constraint is met between items A and B, the full alignment of all three items is infeasible as the sum of the adjacent score widths of items B and C is less than the minimum scoring distance τ; hence the knives are unable to move close enough together to score the lines in the required locations.

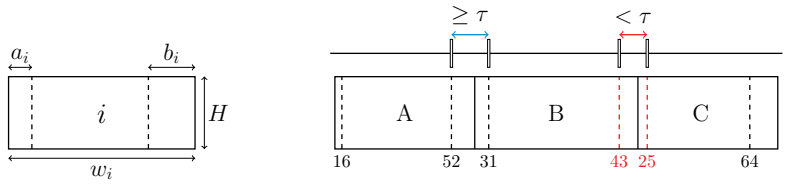


Figure 1: Dimensions of an item i marked with dashed score lines, and an example packing showing both feasible and infeasible alignments of three items to be scored by pairs of knives. Here, the minimum scoring distance $\tau = 70$.

The remainder of this section will formally define both this single-bin problem and the corresponding multi-bin problem, known as the Score-Constrained Packing Problem (SCPP). In the next section, we will give a polynomial-time algorithm that exactly solves the single-bin problem. Section 3 then explains the difficulties associated with the SCPP and analyses existing heuristics from literature. A tailored evolutionary algorithm (EA) for the SCPP is then presented in Section 4, along with results from our experiments. Finally Section 5 concludes the paper and discusses outcomes and possible directions for further work. A summary of the notation used in this paper is provided in Table 1.

1.1. Problem Definitions

We now formally define the main problem to be investigated in this paper:

Definition 1. *Let $\mathcal{I}$ be a set of n rectangular items of height H with varying widths w_i and score widths a_i, b_i for all $i \in \mathcal{I}$. Given a minimum scoring distance τ, 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 (a) the vicinal sum constraint is satisifed in each bin and (b) no bin is overfilled.*

Each item $i \in \mathcal{I}$ can be packed into a bin in either a regular orientation, denoted (a_i, b_i), where the smaller score width a_i is on the left-hand side of item i, or a rotated orientation (b_i, a_i), where the larger score width b_i is on the left-hand side. Therefore, for the SCPP, $\mathcal{F}$ is the set of all item subsets that can be feasibly packed into a bin such that the vicinal sum constraint is fulfilled. Thus, there is the packing sub-problem within each individual bin, defined as follows:

Definition 2. *Let $\mathcal{I}' \subseteq \mathcal{I}$ be a set of rectangular items whose total width $A(\mathcal{I}') = \sum_{i \in \mathcal{I}'} w_i$ is less than or equal to the bin width W. Then, given a minimum scoring distance τ, 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.*

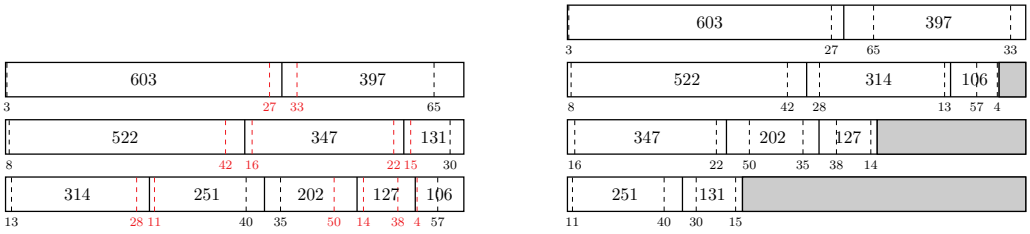
Table 1: Summary of the notation used for the problems considered in this paper.

Term	Description
$\mathcal{I}$	An instance of the problem comprising n items.
W	The maximum capacity of each bin.
$\mathcal{S}$	A feasible solution for an instance of the SCPP comprising bins $S_1, \ldots, S_k$.
$\mathcal{F}$	The set of all item subsets that can be feasibly packed into a single bin.
w_i	The width of an item $i \in \mathcal{I}$.
a_i, b_i	The score widths of an item $i \in \mathcal{I}$, with $a_i \leq b_i$.
τ	The minimum scoring distance.
$A(.)$	The total width/area of the set of items between the parentheses.
$\mathcal{M}$	An instance of the COP, which is a multiset of unordered pairs of integers.
$\mathcal{T}$	A solution for an instance $\mathcal{M}$ of the COP.
V	A vertex set comprising $2n + 2$ vertices.
$w(v_i)$	The weight of a vertex v_i.
$p(v_i)$	The partner of a vertex v_i.
B	The set comprising $n + 1$ "blue" edges between partner vertices.
R	The set comprising "red" edges between vertices that meet the vicinal sum constraint and are not partners.
$m(v_i)$	The match of a vertex v_i.
R'	A set comprising red edges between matched vertices, $R' \subseteq R$.
$C_1, \ldots, C_z$	The cyclic components of the subgraph $G' = (V, B \cup R')$.
$\mathcal{L}$	A list of edges in R'.
$\mathcal{R}''$	A collection of edge subsets $R''_1, R''_2, \ldots$
t	The theoretical minimum $t = \lceil \sum_{i=1}^{n} w_i/W \rceil$, which is a lower bound for k.
$f(\mathcal{S})$	The fitness function $f(\mathcal{S}) = \sum_{S_j \in \mathcal{S}} (A(S_j)/W)^2/\lvert\mathcal{S}\rvert$.

It is this single-bin problem, the sub-SCPP, that was originally introduced as an open-combinatorial problem by Goulimis in 2004 and subsequently studied by Becker (2010), Lewis et al. (2011), and Becker & Appa (2015). However, the only known studies at the time of writing on the problem involving multiple bins (i.e. the SCPP) is by Lewis et al. (2011), Hawa et al. (2018), and Hawa (2020).

In Fig. 1, observe that a feasible alignment of the three items in a single bin can be obtained by rotating item C.[1] However, because there are $2^{n-1}n!$ distinct orderings of n items in a single bin, it is clear that enumerative methods are not suitable. Fig. 2 shows feasible solutions for a set of items $\mathcal{I}$ as an instance of the BPP and the SCPP. For the SCPP, an extra bin is required to accommodate all items whilst fulfilling the vicinal sum constraint. Note that the solution produced for the BPP is not feasible for the SCPP in this case as the constraint is violated at least once in every bin. Thus, the BPP can be seen as a special case of the SCPP when $\tau = 0$ as the vicinal sum constraint will always be satisfied.

[1]Note that the outermost score widths in each bin are disregarded as they are not adjacent to any other items.



(a) BPP (b) SCPP

Figure 2: Solutions for the BPP and SCPP using the same set $\mathcal{I}$ of 10 items and $W = 1000$. For the SCPP, $\tau = 70$. The red score lines on the solution for the BPP show the vicinal sum constraint violations if it were to be used as a solution for the SCPP.

The BPP forms the basis of many packing problems including those of different item and bin shapes, sizes, and dimensions: Haouari & Serairi (2009) and Liu et al. (2021) both studied the variable-sized bin packing problem using bins of unequal sizes, whilst Zhou et al. (2009) extended the problem such that, in addition to differing bin sizes, items and bins are assigned different types; bins can only accommodate items of the same type. Problems comprising circular items have been addressed by Yuan et al. (2021) and He et al. (2021), where the items must be packed into a minimum number of circular and square bins respectively. One particular problem of interest to us involving non-rectangular items is the trapezoid packing problem (TPP) (Lewis et al., 2011; Lewis & Holborn, 2017), where trapezoids 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.

Another variation of the 2D-BPP requires rectangular items to be packed in their given orientations (Lodi et al., 1999; Ntene & van Vuuren, 2008), whilst much literature exists on the problem where items can be rotated by 90° (Kenmochi et al., 2009; Cui et al., 2013; He et al., 2013). Other problems related to the BPP include guillotineable constraints (Bennell et al., 2018; Kröger, 1995; Hifi, 1998), shelf divisions (Xavier & Miyazawa, 2008), and unloading constraints (da Silveira et al., 2013), as well as the dual version of the BPP studied by Csirik & Totik (1988) in which the aim is to pack the items into the maximum number of bins such that the capacity in each bin is at least some specified value C.

Another problem similar to the BPP is the cutting stock problem (CSP), which involves cutting large pieces of material into smaller pieces whilst minimising material wasted. The main difference between the BPP and the CSP is that, according to the typology of Wäscher et al. (2007), the items in the BPP tend to be strongly heterogeneous (the items are of many different sizes) whilst in the CSP the items are weakly heterogeneous (many items have the same size). The classical CSP has been widely studied, with notable works by Gilmore & Gomory (1961, 1963), however, as with the BPP, many adaptations stem from the CSP to satisfy real-life problems. Similarly to above, Belov & Scheithauer (2002) and Poldi & Arenales (2009) consider the CSP with varying stocks lengths with the aim of reducing waste materials. Furthermore, with the increase in interest in sustainability, Coelho et al. (2017) and do Nascimento et al.

(2020) focus on solving the CSP such that any leftover materials produced can be used in the future or sold for a profit. Cui et al. (2017) considers the same problem but using previous leftover materials along with new material in subsequent orders. In addition, there exists an abundance of literature concerning the reduction of initial setup costs: Wu & Yan (2016) propose a balance approach, whilst Ma et al. (2019) implement two heuristics to address the significant setup times for the multi-period capacitated CSP, where multiple rods can also be cut simultaneously.

One particular case of the CSP, described by Garraffa et al. (2016), considers sequence-dependent cut-losses (SDCL). Here, rectangular items of varying lengths are to be cut from strips of material of fixed lengths; however the type of cutting machine used results in material loss between items during the cutting process. The amount of loss can vary between different items, and is also dependent on the order of the items (i.e. 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). Hence, the CSP-SDCL involves packing the items into the fewest number of bins such that the sum of item lengths *and* the sum of cut losses between all adjacent items in each bin does not exceed the bin capacity.

As with the TPP and CSP-SDCL, the SCPP not only involves deciding which bin each item should be packed into, but also, unlike the BPP, *how* the items should be packed – that is, determining the order and orientation of items within each bin. One specific difference, however, concerns the feasibility of individual bins. In the TPP, although clearly not optimal, it is still legal to place trapezoids with opposite angles, i.e. '\' and '/', alongside one another. Likewise in the CSP-SDCL, two items with a large cut loss between them can still be packed alongside one another if necessary. Both of these problems allow items to be packed in *any* order and orientation as long as the bins are not overfilled. In contrast, the SCPP possesses the strong vicinal sum constraint which, if violated, immediately causes an alignment of items in a bin to be invalid, thus rendering the entire solution infeasible. It is this distinction that leads us to seek new methods capable of producing high quality solutions that fulfil the constraints of the SCPP in a reasonable amount of time.

2. Solving the Score-Constrained Packing Sub-Problem

In this section we focus on the sub-SCPP, which involves packing items into a single bin. To begin, consider the following sequencing problem defined by Hawa et al. (2018):

Definition 3. *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 element is an ordered pair. 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 ordered pairs is greater than or equal to τ.*

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

$(61, 48), (26, 13), (57, 44), (32, 39), (35, 41), (29, 17)\rangle$. It is evident that the COP is in fact equivalent to the sub-SCPP, whereby each pair in $\mathcal{M}$ can be seen as an item i represented by its score widths a_i, b_i, and the value τ is the minimum scoring distance. It follows that the requirement for the sum of adjacent values to equal or exceed τ corresponds to the vicinal sum constraint (2). The items in an instance $\mathcal{I}'$ of the sub-SCPP have total width $A(\mathcal{I}') \leq W$, and so when seeking a feasible arrangement the items' widths can be disregarded. Therefore, we are able to simplify any given instance $\mathcal{I}'$ of the sub-SCPP by transforming $\mathcal{I}'$ into an instance of the COP.

In this section we present the Alternating Hamiltonian Construction (AHC) algorithm, a polynomial-time algorithm for solving the COP and hence also the sub-SCPP. The underlying algorithm was originally proposed by Becker (2010) and determines whether a feasible solution exists for a given instance. This is extended here so that, if a solution does indeed exist, AHC is able to construct the final solution quickly. In addition, we also simplify and increase the efficiency of this algorithm.

To prevent executing the algorithm unnecessarily, a basic preliminary test is first performed. Of the $2n$ values in $\mathcal{M}$, suppose the two smallest values are placed in the outermost positions in the sequence $\mathcal{T}$. Then, if the third smallest value in $\mathcal{M}$ and the largest value in $\mathcal{M}$ do not sum up to greater than or equal to τ there cannot exist a feasible ordering of all elements in $\mathcal{M}$. As an example, consider the instance $M = \{\{1, 46\}, \{3, 52\}, \{8, 30\}, \{2, 61\}\}$ and $\tau = 70$. The third smallest value, 8, and the largest value, 61, do not add up to τ, and so as there is no larger value in $\mathcal{M}$ that can be aligned alongside 8 in the sequence $\mathcal{T}$, it is not possible for a feasible solution to exist. Note that a positive outcome from this test does not necessarily imply that a feasible solution exists for the instance; however a negative outcome confirms the non-existence of a solution.

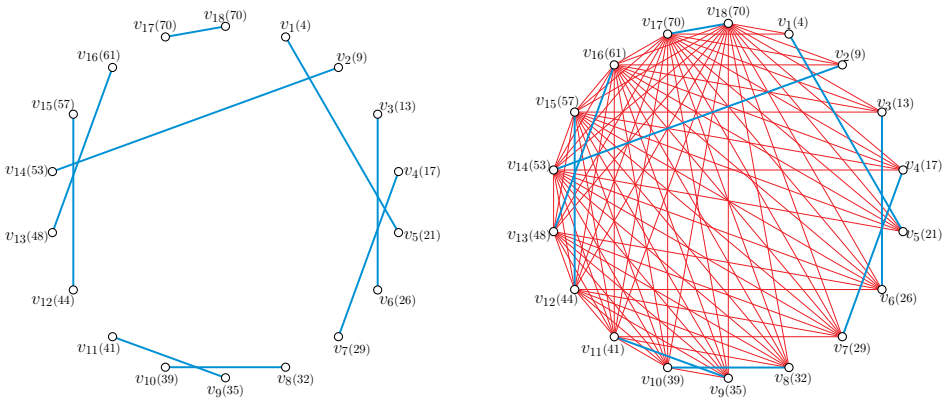
2.1. Modelling the Constrained Ordering Problem

If an instance $\mathcal{M}$ of the COP has passed the preliminary test and has not yet been deemed infeasible, we can proceed to model $\mathcal{M}$ graphically. For each pair $\{a_i, b_i\} \in \mathcal{M}$, two vertices u, v with weights $w(u) = a_i$, $w(v) = b_i$ are created, together with a "blue" edge $\{u, v\}$. Vertices joined by a blue edge are referred to as *partners*. This gives a vertex-weighted graph G comprising n components. 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} is then added to G with weights $w(v_{2n+1}) = w(v_{2n+2}) = \tau$, together with a blue edge $\{v_{2n+1}, v_{2n+2}\}$. All blue edges between partners are now said to belong to the edge set B. It is also useful to denote the partner of a vertex v_i as $p(v_i)$; thus the set B can be written as $\{\{v_i, p(v_i)\} : v_i \in V\}$. Note that $|B| = n + 1$, so B is a perfect matching. Fig. 3a depicts the graph $G = (V, B)$ comprising $n + 1$ components based on the example instance $\mathcal{M}$ of the COP stated at the beginning of this section.

Next, a second set of "red" edges, R, is added to G containing edges between vertices that are not partners and whose combined weight equals or exceeds τ; thus $B \cap R = \emptyset$. Fig. 3b illustrates the resulting graph $G = (V, B \cup R)$ produced from our example instance

$\mathcal{M}$. The graph has a noticeable pattern, with the degree of each vertex increasing in accordance with the weight of the vertices. It can also be seen that 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 their weights mean they are adjacent to every other vertex via an edge in R.



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

Figure 3: The graph $G = (V, B \cup R)$ modelling our example instance $\mathcal{M}$ of the COP. Members of B are shown by thick blue edges and R by thin red edges, with the vertices' weights in parentheses.

Now, recall that a Hamiltonian cycle in a graph G is a cycle that visits every vertex of G exactly once. A graph containing such a cycle is said to be Hamiltonian. From this, we present the following definition:

Definition 4. *Let $G = (V, B \cup R)$ be a simple, undirected graph where each edge is a member of exactly one of two sets, B or R. An* alternating Hamiltonian cycle *in G is a Hamiltonian cycle whose successive edges alternate between sets B and R.*

All values in $\mathcal{M}$ must be in the solution $\mathcal{T}$, and so all the vertices on our graph G must be in a single cycle that represents a feasible solution; thus we require a Hamiltonian cycle. The order of the values within each pair in $\mathcal{M}$ can be rearranged, however the values themselves cannot be changed – every pair of values in $\mathcal{M}$ must remain as a pair in the final sequence $\mathcal{T}$. It follows then that we need a Hamiltonian cycle in which each vertex is either preceeded by or succeeded by its partner vertex in the cycle. Therefore, our aim is to seek an alternating Hamiltonian cycle in the graph G.

Observe that an alternating Hamiltonian cycle in G corresponds to a legal sequence of the elements in $\mathcal{M}$ because (a) the edges in B represent each pair of values in $\mathcal{M}$, and (b) edges from R depict the values that meet the vicinal sum constraint (Hawa, 2020). As all edges in B must be in the final cycle, the task involves finding a suitable matching subset of red edges $R' \subseteq R$ that, together with the blue edges in B, form an alternating Hamiltonian cycle in G. The universal vertices, v_{2n+1} and v_{2n+2}, aid the construction of the alternating Hamiltonian cycle as they are able to connect to the lowest-weighted vertices that correspond to the values in $\mathcal{M}$ that will be in the outermost positions of the sequence $\mathcal{T}$; however once a cycle has been produced these vertices and any incident

edges are removed, resulting in a path corresponding to a feasible COP solution $\mathcal{T}$. Fig. 4 shows the alternating Hamiltonian cycle found in G for our example instance of the COP, which translates to a feasible solution $\mathcal{T}$.

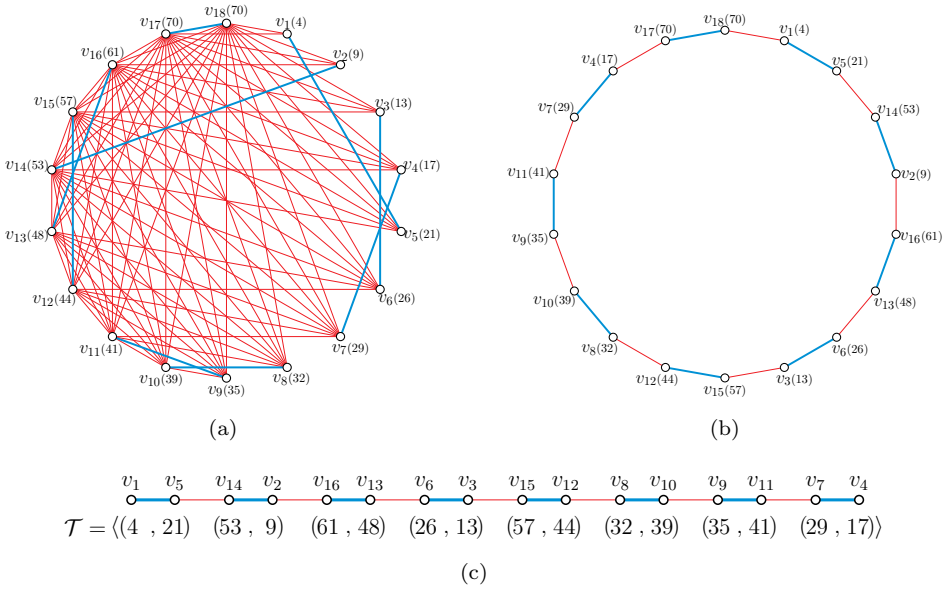


Figure 4: (a) The graph $G = (V, B \cup R)$ modelling our example instance $\mathcal{M}$; (b) a subgraph of G comprising all edges in B and a subset of edges $R' \subseteq R$ that form an alternating Hamiltonian cycle; and (c) removing the universal vertices produces an alternating path corresponding to a feasible solution $\mathcal{T}$.

In general, determining whether a graph is Hamiltonian is NP-complete, whilst the problem of actually finding a Hamiltonian cycle is NP-hard (Karp, 1972). Consequently, the alternating Hamiltonian cycle problem is also NP-hard, as it is a generalisation of the former (Häggkvist, 1977). Despite this, due to the special structure of the graphs modelling instances of the COP, here we are able to determine the existence of an alternating Hamiltonian cycle in polynomial-time (Hawa et al., 2018).

2.2. The Alternating Hamiltonian Construction Algorithm

Our algorithm for finding an alternating Hamiltonian cycle in G is the Alternating Hamiltonian Construction (AHC) algorithm. AHC comprises two subprocedures: one to produce an initial matching $R' \subseteq R$, and another to modify R', if necessary, so that it contains suitable edges that form an alternating Hamiltonian cycle with the fixed edges B in G. We now describe these two stages.

2.2.1. Finding a matching $R' \subseteq R$

The first subprocedure of AHC is the Maximum Cardinality Matching (MCM) algorithm, which seeks a matching R' comprising $n + 1$ edges on the graph induced by the set of red edges R. Note that this could actually be achieved via standard matching processes such as the Blossom algorithm (Edmonds, 1965); however due to the special structure of

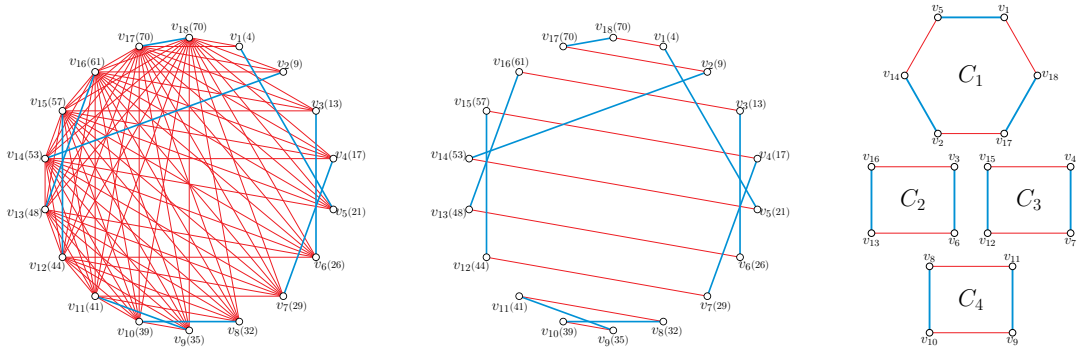
G, such a matching can also be achieved via more efficient methods (Mahadev & Peled, 1994; Becker & Appa, 2015).

Algorithm 1 MCM ($G = (V, B \cup R)$)

1: $R' \leftarrow \emptyset$
2: $m(v_i) \leftarrow$ NULL $\forall\ v_i \in V$
3: **for** $i \leftarrow 1$ **to** $2n+2 : m(v_i) =$ NULL **do**
4: **for** $j \leftarrow 2n+2$ **to** $i+1 : m(v_j) =$ NULL **do**
5: **if** $\{v_i, v_j\} \in R$ **then**
6: $m(v_i) \leftarrow v_j$, $m(v_j) \leftarrow v_i$
7: $R' \leftarrow R' \cup \{\{v_i, v_j\}\}$
8: **break**
9: **if** $m(v_i) =$ NULL **and** $i \neq 1$ **and** $m(v_{i-1}) \neq$ NULL **and** $m(p(v_i)) =$ NULL **and** $\{v_{i-1}, p(v_i)\} \in R$ **then**
10: $R' \leftarrow R' \backslash \{\{v_{i-1}, m(v_{i-1})\}\}$
11: $m(v_i) \leftarrow m(v_{i-1})$, $m(m(v_i)) \leftarrow v_i$
12: $m(v_{i-1}) \leftarrow p(v_i)$, $m(p(v_i)) \leftarrow v_{i-1}$
13: $R' \leftarrow R' \cup \{\{v_{i-1}, p(v_i)\}\} \cup \{\{v_i, m(v_i)\}\}$
14: **return** R'

As shown in the pseudocode in Algorithm 1, vertices are considered in turn in weight-ascending order and are matched with the highest-weighted unmatched vertex adjacent in R. Here, the *match* of a vertex v_i is denoted as $m(v_i)$. The set R' then consists of all edges from R between matched vertices, i.e. $\{\{v_i, m(v_i)\} : v_i \in V\}$. In the event that a vertex v_i is not adjacent to any other vertex via an edge in R, the previous vertex v_{i-1} can be rematched provided v_{i-1} has been matched successfully and is adjacent to v_i's partner, $p(v_i)$. Then, v_i is matched with v_{i-1}'s match, and v_{i-1} is rematched with $p(v_i)$ (Lines 9–13).

It has been proven that MCM returns a matching R' of maximum cardinality; thus, on completion of MCM, if R' does not contain $n+1$ edges then there are too few edges in R' to form an alternating Hamiltonian cycle with the edges in B (Hawa, 2020). Consequently, no feasible solution can exist for the given instance $\mathcal{M}$ of the COP, and computation can terminate. On the other hand, if $|R'| = n+1$ then MCM has successfully produced a perfect matching, and the spanning subgraph $G' = (V, B \cup R')$ will be a 2-regular graph consisting of cyclic components $C_1, C_2, \ldots, C_z$, where 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'. Clearly, if $z = 1$ then G' is an alternating Hamiltonian cycle and a solution has been found; otherwise, G' comprises multiple cycles and AHC must find a way of connecting these components together to form a single alternating Hamiltonian cycle. Fig. 5 shows the subgraph G' formed using the matching R' procured by MCM. In Fig. 5b we also see the pattern of the red edges in R', connecting the lowest-weighted vertices to the highest-weighted vertices. By depicting G' in planar form as in Fig. 5c, it is clear that G' comprises $z = 4$ cyclic components.



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

Figure 5: (a) The graph G modelling an example instance $\mathcal{M}$; (b) the subgraph $G' = (V, B \cup R')$ with the perfect matching $R' \subseteq R$ found by MCM; and (c) a planar embedding of G' showing $z = 4$ cyclic components.

2.2.2. Modifying the matching $R' \subseteq R$

To merge the components of G' into a single alternating Hamiltonian cycle, we need to remove edges from each cyclic component of G' and replace these edges with different edges that join the components together. Edges in B cannot be removed from G' as all edges in B must be in the final cycle; we therefore need to modify the matching R'. This task involves (a) deciding which edges to remove from R' that break apart the cyclic components of G', leaving vertices that are only incident to their partners via a blue edge in B, and then (b) determining which edges from $R \backslash R'$ to add to R' that can connect the vertices of different components together.

The second subprocedure of AHC is the Bridge-Cover Recognition (BCR) algorithm, based on a method by Becker (2010). BCR seeks to identify a collection $\mathcal{R}''$ of distinct subsets of edges in different components of G' that will be removed from R'. These edge subsets will also be used to identify the new edges from $R \backslash R'$ to be added to R' that will act as bridges, connecting those components into a single component. Here, if an edge from a component C_j of G' is in a subset in the collection $\mathcal{R}''$, then $\mathcal{R}''$ is said to *cover* the component C_j. Clearly, to connect all components of G' together, at least one edge must be removed from each component. BCR aims to produce 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-weighted vertices of the edges are in ascending order (see Fig. 6a). Then, in 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-weighted vertex of each edge is adjacent to the higher-weighted vertex of the next edge via an edge in $R \backslash R'$;

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

(iii) only one edge is in a component already covered by $\mathcal{R}''$; 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 the search for another subset of edges.[2] Once the penultimate edge in $\mathcal{L}$ has been assessed, edges in $\mathcal{R}''$ are removed from $\mathcal{L}$ and the next iteration begins. BCR ends the search successfully once $\mathcal{R}''$ covers all z components of G'. However, if no new subsets are created during an iteration, or if fewer than two edges remain in $\mathcal{L}$ after an iteration and $\mathcal{R}''$ does not cover all z components, then no more subsets exist and it can be concluded with certainty that no feasible solution exists for the given instance of the COP (Hawa, 2020). Fig. 6 shows the BCR process on our example instance, where the subsets $R_1'' = \{\{v_2, v_{17}\}, \{v_3, v_{16}\}, \{v_4, v_{15}\}\}$ and $R_2'' = \{\{v_7, v_{12}\}, \{v_8, v_{11}\}\}$ have been formed. As $\mathcal{R}'' = \{R_1'', R_2''\}$ covers all four components of G', no more subsets are required.

If BCR has been able to acquire a feasible collection $\mathcal{R}''$ covering all z components of G', then there exists an alternating Hamiltonian cycle in G. The subsets in $\mathcal{R}''$ contain the edges to be removed from R', and also indicate the new edges from $R \backslash R'$ to add to R' that will connect the components together. BCR uses each subset $R_i'' \subset \mathcal{R}''$ to procure the replacement edges from $R \backslash R'$ as follows: for each edge in R_i'' in turn, the edge from $R \backslash R'$ connecting the lower-weighted vertex of the edge to the higher-weighted vertex of the next edge is added to R'. These edges form bridges between vertices of different components (as shown in Fig. 6b). The edges in $\mathcal{R}''$ are then removed from R', so that R' remains a perfect matching of cardinality $n + 1$. The resulting graph $G' = (V, B \cup R')$ depicted in Fig. 6c using the modified matching R' is an alternating Hamiltonian cycle. Removing the universal vertices yields an alternating path which corresponds to a feasible solution $\mathcal{T}$ (Fig. 6d).

In the first incarnation of this algorithm (Becker, 2010), a procedure was proposed that searches through $\mathcal{L}$ just once to find edge subsets for the collection $\mathcal{R}''$. However, for some instances, although $\mathcal{R}''$ covers all components of G', these components cannot be connected into a single alternating Hamiltonian cycle. This issue stems from the requirements for edges to form a subset, where this previous procedure allowed edges to be in multiple components already covered by $\mathcal{R}''$. Therefore, we introduce Condition (iii), which permits only *one* edge in a new subset R_i'' to be in a component covered by $\mathcal{R}''$. This prevents unnecessary additional edges from being included in R', ensuring that the components are linked to produce a single cycle. Fig. 7 shows the formation of $\mathcal{R}''$ using the original procedure, where the subset R_2'' contains edges in *two* components that $\mathcal{R}''$ already covers. Although $\mathcal{R}'' = \{R_1'', R_2''\}$ covers all components of G', the bridges obtained from these subsets link C_2 and C_3 twice, thus connecting the four components into two different components. Therefore, we replaced the initially proposed procedure with BCR, which rectifies the issue and operates in a more efficient manner.

This concludes the Alternating Hamiltonian Construction (AHC) algorithm. The pseudocode for the entire procedure is provided in Algorithm 2 which returns, for any graph G modelling an instance $\mathcal{M}$ of the COP (as described at the beginning of this section), an alternating Hamiltonian cycle in G if one exists.

[2] When searching for edges to produce the first subset, R_1'', only Conditions (i) and (ii) are required since $\mathcal{R}'' = \emptyset$.

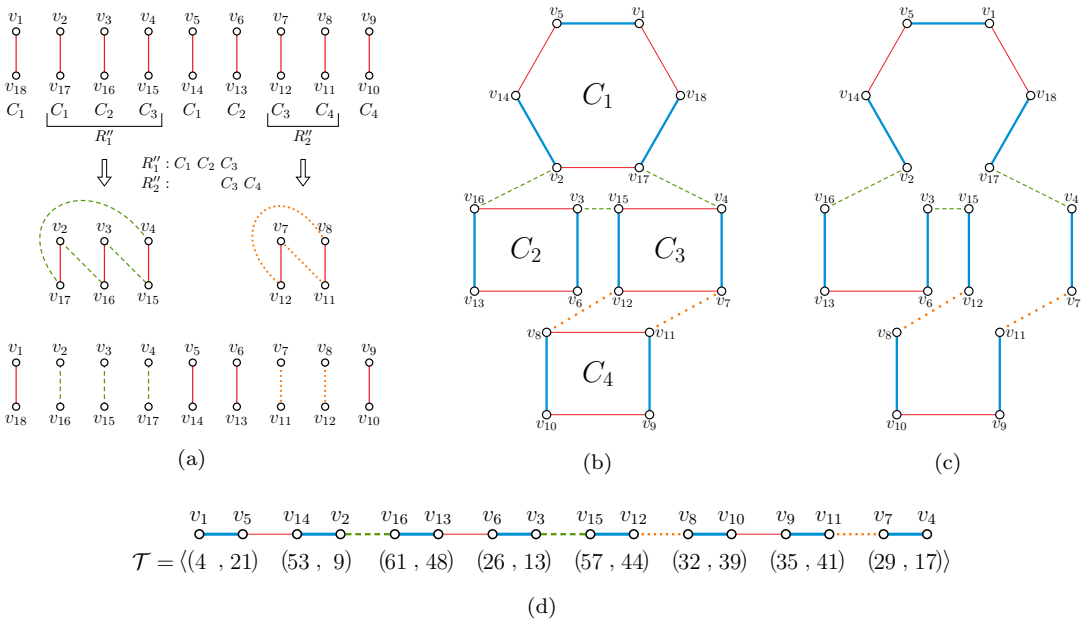


Figure 6: BCR creates a collection $\mathcal{R}'' = \{R''_1, R''_2\}$ of subsets containing edges in R' that when replaced by edges from $R \backslash R'$ connects the components of G' into a single alternating Hamiltonian cycle. Dashed green edges and dotted orange edges are the bridges from R''_1 and R''_2 respectively. The resulting alternating path corresponds to a solution $\mathcal{T}$. Note that (c) is the same cycle shown in Fig. 4b.

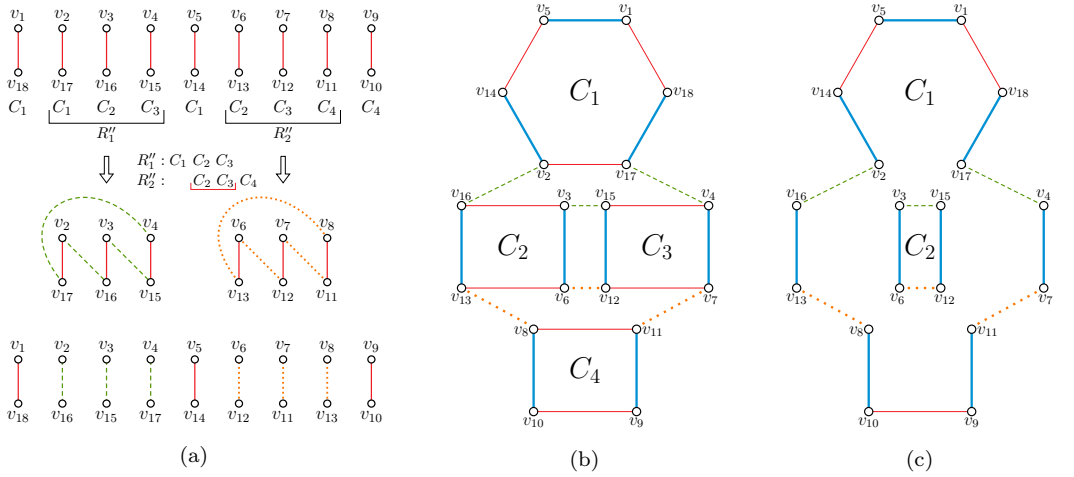


Figure 7: The procedure proposed by Becker (2010) creates subsets in $\mathcal{R}''$ each containing edges in both C_2 and C_3, resulting in two cyclic components in G' as opposed to a single alternating Hamiltonian cycle.

For any instance $\mathcal{M}$ of the COP, our 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 1. *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 $\lfloor \frac{n+1}{2} \rfloor$ 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 $\lfloor \frac{n+1}{2} \rfloor - 1$. 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 2 AHC $(G = (V, B \cup R))$

$R' \leftarrow$ MCM$(G = (V, B \cup R))$
if $|R'| < n + 1$ **then**
    not enough edges to form an alternating Hamiltonian cycle
    infeasible, **end**
$G' = (V, B \cup R')$ comprises z cyclic components
if $z = 1$ **then**
    G' is an alternating Hamiltonian cycle
    feasible, **end**
$\mathcal{R}'' \leftarrow$ 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
    no suitable subset of edges $R' \subseteq R$ exists that can connect the components
    infeasible, **end**
return either alternating Hamiltonian cycle or statement of infeasibility

As the AHC algorithm can solve instances of the COP, it follows that this polynomial-time exact algorithm can also solve all instances of the sub-SCPP. That is, given any set of items, we are able to find a feasible arrangement (if one exists) using AHC and pack the items in the correct order and orientation into a bin.

3. Heuristics for the Score-Constrained Packing Problem

We now consider the multi-bin version of the sub-SCPP, the Score-Constrained Packing Problem (SCPP) described in Section 1, in which a set $\mathcal{I}$ of n items are to be partitioned into a set of bins $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ according to Constraints (1a)–(1c). Recall here that a bin $S_j \in \mathcal{F}$ if and only if the total width of items in the bin, $A(S_j) = \sum_{i \in S_j} w_i$,

does not exceed the bin's capacity W and the vicinal sum constraint (2) is fulfilled. An optimal solution for the SCPP is a solution comprising the fewest number of bins required to feasibly pack all items in $\mathcal{I}$. The aim is to therefore minimise the number of bins k.

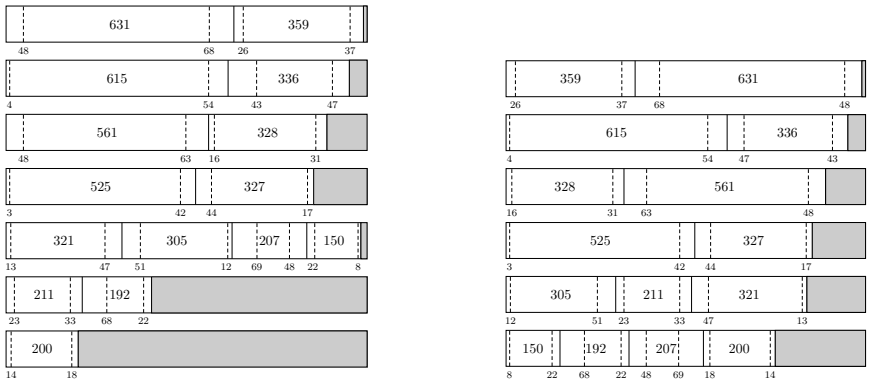
The BPP is known to be NP-hard (Garey & Johnson, 1979), and since the SCPP generalises the BPP it follows that the SCPP is also NP-hard. Assuming P≠NP, we therefore cannot hope to find an optimal solution for all instances of the SCPP in polynomial-time. A well-known heuristic for the BPP is First-Fit (FF), a greedy algorithm that packs each item one by one in some given order into the lowest-indexed bin such that the capacity of the bin is not exceeded. It is known that there always exists at least one ordering of the items such that FF produces an optimal solution, though identifying such an ordering is itself NP-hard (Lewis, 2009). An improvement on FF is the First-Fit Decreasing (FFD) heuristic, where items are considered in non-increasing order of size. It has been proven that the worst case for FFD is $\frac{11}{9}k + \frac{6}{9}$, and that this bound is tight (Dósa, 2007). Similar heuristics include Best-Fit (BF), in which each item is packed into the fullest bin that can accommodate the item without being overfilled, and its offline counterpart Best-Fit Decreasing (BFD). A comprehensive overview of these heuristics and related methods can be seen in Coffman et al. (1984). More advanced heuristics for the BPP have also been developed, such as the Minimum Bin Slack (MBS) heuristic (Gupta & 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 & Hindi (2002).

For the BPP, a basic lower bound for k is the theoretical minimum, $t = \lceil \sum_{i=1}^{n} w_i / W \rceil$ (Martello & Toth, 1990). Note, however, that t will not perform as accurately for the SCPP as it fails to account for the vicinal sum constraint. For example, given a set of n items in which the largest score width $b_i < \tau/2$, it is clear that no pairs of score widths can fulfil the vicinal sum constraint, meaning that any feasible solution will feature n bins.

The vicinal sum constraint also introduces further differences between the BPP and SCPP. The obvious disparity is that of the ordering and orientation of the items in the bins: unimportant in the BPP, but vital for the feasibility of a solution for the SCPP. Another distinction arises when attempting to modify solutions. In the BPP, a bin remains feasible when an item is removed or a new item is added (provided the bin can accommodate the item), whereas for the SCPP this may render a bin infeasible as the new set of score widths may not abide by the vicinal sum constraint. Consequently, heuristics for the BPP will need to be adapted in order to produce feasible solutions for the SCPP.

It is worthwhile mentioning that other lower bounds exists in the literature for the BPP (see, for example, Martello & Toth (1990) and Chan et al. (1998)); however, due to the novelty of the SCPP we opted to analyse solutions to the problem using the basic lower bound t for the BPP, as currently there does not exist a lower bound specifically for the SCPP. Note that using a different lower bound would not alter the interpretation of the strengths and weaknesses of different algorithms for the SCPP.

As the SCPP is a relatively new problem, few methods have been seen in literature. Some basic heuristics were introduced by Hawa et al. (2018), two of which are based on the FFD heuristic for the BPP. The Modified First-Fit Decreasing (MFFD) heuristic performs in the same fashion as FFD with the additional step that an item i can only be packed into a bin S_j if the score width on the end of S_j and one of i's score widths, a_i or b_i, meet the vicinal sum constraint. An advancement of this heuristic is MFFD$^+$, which incorporates the entire AHC algorithm. Rather than attempting to pack an item i into the ends of a bin S_j, MFFD$^+$ calls upon AHC to find a feasible ordering of all items in S_j together with item i, i.e. AHC is used to determine the membership of $\mathcal{F}$. Clearly, MFFD$^+$ is the superior of the two, as the application of AHC guarantees that a feasible configuration of items in a bin will be found if it exists. To demonstrate this, Fig. 8 compares solutions produced using MFFD and MFFD$^+$ for the same instance of the SCPP. Note that in this case MFFD$^+$ has formed an optimal solution as it comprises $t = 6$ bins.



(a) MFFD (b) MFFD$^+$

Figure 8: Solutions formed using the MFFD and MFFD$^+$ heuristics. Here, $|\mathcal{I}| = 15$, $W = 1000$, and $\tau = 70$. As the theoretical minimum $t = 6$ in this case, MFFD$^+$ has produced an optimal solution.

4. An Evolutionary Algorithm for the Score-Constrained Packing Problem

In this section we introduce an evolutionary algorithm (EA) for the SCPP in order to improve on the heuristics discussed in the previous section. An EA is a metaheuristic algorithm inspired by evolution via natural selection. A set of candidate solutions to the problem forms the initial population, and procedures emulating selection, reproduction, recombination and mutation are then used to create new "offspring" solutions.

Clearly, in this problem the set of feasible packings $\mathcal{F}$ will be too large to enumerate in all but the most trivial of instances. EAs are a suitable alternative to enumeration and have been found to produce good results for a variety of grouping problems (Lewis & Holborn, 2017; Falkenauer, 1996; Quiroz-Castellanos et al., 2015). Here, we investigate three different group-based recombination operators within an EA framework which includes a local search procedure inspired by Martello & Toth (1990). The AHC algorithm

described in Section 2 is also integrated into the EA to solve instances of the sub-SCPP as and when they occur.

4.1. Representation

In EAs, solutions to the problem are often represented or "encoded" using strings of characters, integers, or binary values referred to as *chromosomes*, with the *genes* of each chromosome relating to the individual components of the solution. The most obvious encoding for grouping problems consists in assigning one gene per element: for example, the chromosome (1,3,4,1,2,4) would represent a solution where the first and fourth elements are in group 1, the fifth element is in group 2, the second element is in group 3 and the third and sixth elements are in group 4. However, this type of solution encoding contradicts the Principle of Minimal Redundancy (Radcliffe et al., 1991) whereby each possible feasible solution to the given problem instance should be represented by the fewest number of distinct chromosomes so that the overall size of the search space is reduced. It can be seen that the solution represented by the above chromosome can be encoded using another entirely different chromosome by relabelling the groups, for example (3,4,2,3,1,2). It then follows that, using this encoding scheme, a solution containing k groups can be represented by $k!$ distinct chromosomes.

Although there are various other encoding schemes for grouping problems (see, for example, Falkenauer (1993)), in the SCPP it is crucial to know the ordering and orientation of the individual items within each bin of a solution. Consequently, in our EA framework we continue using the description of a solution $\mathcal{S}$ for an instance $\mathcal{I}$ of the SCPP (see Section 3). Similar approaches have been used by Galinier & Hao (1999) and Lewis & Holborn (2017).

4.2. Recombination

Recombination is used in EAs to generate new solutions by taking existing parent solutions and combining parts of them to create offspring solutions. A recombination operator determines which elements from each parent should be inherited by the offspring. The operators implemented in our EA start with a single offspring $\mathcal{S} = \emptyset$, and use two parent solutions, $\mathcal{S}_1$ and $\mathcal{S}_2$, to build the offspring solution. These operators are designed to ensure all bins in the offspring are feasible, though they may result in a partial offspring solution. In such cases, a repair procedure (described below) is used to re-establish a full solution $\mathcal{S}$.

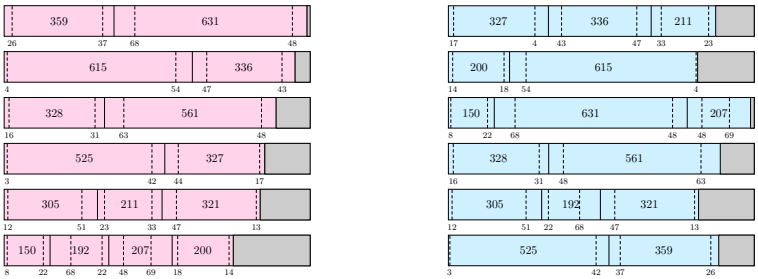
Our first operator is based on the grouping genetic algorithm (GGA) of Falkenauer & Delchambre (1992). First, the bins of the second parent solution $\mathcal{S}_2$ are permuted, and two bins S_i and S_j are selected randomly (where $1 \leq i < j \leq |\mathcal{S}_2|$). All bins between and including S_i and S_j are then copied into the offspring $\mathcal{S}$. Finally, GGA adds to the offspring all bins from $\mathcal{S}_1$ that do not contain any items already present in the offspring.

The second operator we implemented is the alternating grouping crossover (AGX), which is similar to that of Quiroz-Castellanos et al. (2015) proposed for the BPP. Starting with the parent solution containing the fullest bin (breaking ties randomly), AGX copies this bin into the offspring $\mathcal{S}$. Then, bins containing any items in $\mathcal{S}$ are removed

from both $\mathcal{S}_1$ and $\mathcal{S}_2$. The operator then proceeds by copying the fullest bin from the other parent solution into $\mathcal{S}$, and bins are removed from both parents as before. AGX continues to alternate between parents, selecting the fullest bin from each one until at most $\min(|\mathcal{S}_1|, |\mathcal{S}_2|) - 1$ bins have been added to the offspring solution.

Our final operator, AGX′, behaves in a similar manner to AGX; however rather than choosing the fullest bin to copy into the offspring, AGX′ selects bins containing the most items. Both AGX and AGX′ are based on the observation that in high quality solutions, many of the bins will be well-filled. Thus, by selecting bins which are fuller or contain more items from parents to copy into the offspring $\mathcal{S}$, there is the potential to reduce the number of bins in $\mathcal{S}$, as fewer additional bins will be required to pack the remaining items during the repair procedure.

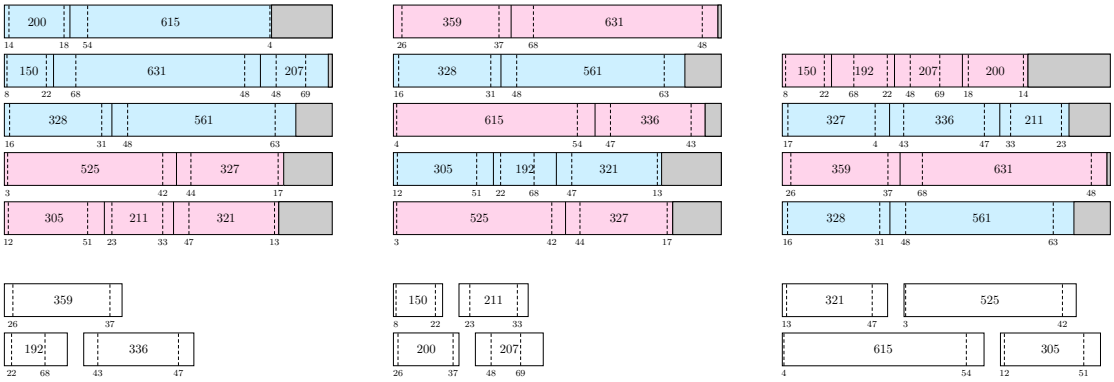
As explained in the previous section, removing an item from a bin in a solution for the SCPP may result in a violation of the vicinal sum constraint. Therefore, in order to maintain feasibility of each bin, the operators need to disregard entire bins in the parent solutions that contain items already in the offspring, rather than removing individual items. These excluded bins, however, may also hold items that are not yet present in the offspring $\mathcal{S}$. Consequently, on completion of the recombination, $\mathcal{S}$ will be a partial solution. To rectify this, the following repair procedure is implemented: first, the MFFD$^+$ heuristic described in Section 3 is applied using just the missing items to form a second partial solution $\mathcal{S}'$. Then, both $\mathcal{S}$ and $\mathcal{S}'$ are used as input into a local search procedure, which produces a full feasible offspring solution. Fig. 9 shows the partial offspring $\mathcal{S}$ produced from two parent solutions using each of the recombination operators, along with the individual items missing from each offspring.



4.3. Local Search

As previously mentioned, our local search method takes two partial solutions $\mathcal{S}$ and $\mathcal{S}'$ containing bins that, together, form a full solution containing all items in $\mathcal{I}$. The aim of this procedure is to strategically shuffle items between the bins of each partial solution. Specifically, we seek to increase the fullness $A(S_j)$ of bins $S_j \in \mathcal{S}$ while maintaining or decreasing the number of items in these bins. As a result, items moved into $\mathcal{S}'$ will be smaller and therefore easier to repack into bins in $\mathcal{S}$ later. The procedure begins by permuting the bins in $\mathcal{S}$ and $\mathcal{S}'$, before attempting to exchange items in four stages:

(1) swapping a pair of items from a bin in $\mathcal{S}$ with a pair of items from a bin in $\mathcal{S}'$;



(a) GGA (b) AGX (c) AGX′

Figure 9: Partial offspring solutions and missing items created from parent solutions $\mathcal{S}_1$ and $\mathcal{S}_2$ using our different recombination operators. In (a) GGA has copied bins S_2, S_3 and S_4 from $\mathcal{S}_2$ to the offspring, and in (b) and (c) $\mathcal{S}_1$ is the initial parent for both AGX and AGX′ as it contains both the fullest bin and the bin with the most items.

(2) swapping a pair of items from a bin in $\mathcal{S}$ with an individual item from a bin in $\mathcal{S}'$;
(3) swapping individual items from bins in $\mathcal{S}$ and $\mathcal{S}'$; and
(4) moving an item from a bin in $\mathcal{S}'$ to a bin in $\mathcal{S}$.

Note that during Stages (1)–(3), the width of the item(s) from $\mathcal{S}'$ must exceed the width of the item(s) from $\mathcal{S}$. In each stage, AHC is executed on the modified groups of items in each bin, and the items are permanently moved only if AHC finds feasible packings for both bins; thus, this local search procedure uses hill-climbing to improve solutions. Other techniques such as simulated annealing and tabu search could also be implemented, however the advantage our using our local search procedure is the quick termination at a local optimum. Once an exchange occurs, the procedure immediately proceeds to the next stage. This process is repeated until all four stages have been conducted in succession with no changes to $\mathcal{S}$ or $\mathcal{S}'$. At this point, the MFFD$^+$ heuristic is applied to any items remaining in $\mathcal{S}'$, generating a new feasible partial solution $\mathcal{S}''$. Finally, the bins in $\mathcal{S}''$ are moved into $\mathcal{S}$, resulting in a full solution.

This method is based on the dominance criterion of Martello & Toth (1990) for the BPP. Variations of this method can be seen in Falkenauer (1996), Levine & Ducatelle (2004), Lewis (2009), and Lewis & Holborn (2017); however the addition of the vicinal sum constraint will often result in fewer changes than seen in these previous implementations.

4.4. The Evolutionary Algorithm Framework

Our EA for the SCPP begins by producing an initial population, with one solution created using MFFD$^+$ and the rest using MFFR$^+$ (where items are packed in random order). Each solution is then mutated and inserted into the population. The mutation of a solution $\mathcal{S}$ consists of permuting the bins, moving $1 < r < |\mathcal{S}|$ randomly selected

bins from $\mathcal{S}$ into a set $\mathcal{S}'$, and then executing local search on these two partial solutions to produce a full feasible solution $\mathcal{S}$. Each iteration of the EA involves selecting two parent solutions $\mathcal{S}_1$ and $\mathcal{S}_2$ from the population at random, applying a recombination operator to produce an offspring solution $\mathcal{S}$, then finally mutating $\mathcal{S}$ before replacing the least fit of the two parents in the population.

As with other BPP algorithms, we use the following function introduced by Falkenauer & Delchambre (1992) to calculate the fitness of a solution $\mathcal{S}$:

$$f(\mathcal{S}) = \frac{\sum_{S_j \in \mathcal{S}} (A(S_j)/W)^2}{|\mathcal{S}|}. \tag{3}$$

In general, this function assigns higher values to solutions using fewer bins; however, there are exceptions. For the BPP it can be shown that, in all cases, if $|\mathcal{S}_1| < |\mathcal{S}_2|$ then $f(\mathcal{S}_1) > f(\mathcal{S}_2)$. Hence a global optimum for $f(\mathcal{S})$ corresponds to a solution using the minimum number of bins. However, this rule is dependent on solutions not having more than one bin that is less than half-full (Falkenauer, 1998). This is easy to ensure with the BPP, but is not always the case with the SCPP due to the vicinal sum constraint. In our case it is therefore necessary to use $f(\mathcal{S})$ only as a tie-breaker between solutions using the same number of bins.

4.5. Experimental Results - EA

Our EA was executed on a set of problem instances for the SCPP created using a problem instance generator. The set contains two types of problem instances: "artificial", in which the items are strongly heterogeneous; and "real", where items are weakly heterogeneous. 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. All problem instances used in our experiments and the problem instance generator are available online (Hawa, 2019b).

Two different bin sizes, $W = 2500$ and 5000 (also of height $H = 1$) were used in the experiments in order to alter the number of items per bin, and the minimum scoring distance τ was set to 70mm – the industry standard. After preliminary experiments, we settled for an initial population containing 25 candidate solutions. Across all instances, the EA was granted a fixed time limit of 600 seconds. The MFFD$^+$ heuristic described in Section 3 was also executed on the same set of problem instances to provide comparative results. Table 2 displays the results obtained from the 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, 2019a).[3] Note that there are

[3]All experiments were implemented in C++ and executed on Windows machines with Intel Core i5-6500 3.20GHz processors and 8GB of RAM.

no benchmark instances available for the SCPP, and so we compare our results with the lower bound t provided in Section 3.

Table 2: Best solutions obtained from the EA using the GGA, AGX, and AGX′ recombination operators, and from the MFFD$^+$ heuristic. Figures in bold indicate the best results for each instance class. Asterisks indicate statistical significance at $\leq 0.05(^*)$ and $\leq 0.01(^{**})$ levels according to a two-tailed paired t-test and two-tailed McNemar's test for the $|\mathcal{S}|$ and $\%t$ columns respectively. Note that the statistical tests were only performed on the recombination operators.

			GGA		AGX		AGX′		MFFD$^+$	
Type, W	$\|\mathcal{I}\|$	t^a	$\|\mathcal{S}\|^b$	$\%t^c$	$\|\mathcal{S}\|$	$\%t$	$\|\mathcal{S}\|$	$\%t$	$\|\mathcal{S}\|$	$\%t$
a, 2500	100	23.32	23.36 ± 4.6	97.4	$^{**}\mathbf{23.34} \pm 4.5$	$^{**}\mathbf{98.6}$	23.36 ± 4.5	97.2	28.46 ± 10.4	2.6
	500	114.94	116.36 ± 2.2	33.4	116.67 ± 2.2	27.4	$^{**}\mathbf{116.28} \pm 2.2$	**35.0**	132.65 ± 4.9	0.0
	1000	229.44	233.93 ± 1.6	1.5	234.58 ± 1.6	1.3	$^{**}\mathbf{233.73} \pm 1.6$	**2.3**	258.39 ± 3.5	0.0
a, 5000	100	11.92	$^{**}\mathbf{12.27} \pm 10.0$	$^{**}\mathbf{84.5}$	12.31 ± 10.2	82.7	12.32 ± 10.2	82.0	19.88 ± 18.3	0.7
	500	57.72	$^{**}\mathbf{61.91} \pm 4.9$	**6.0**	62.44 ± 5.1	4.5	62.50 ± 5.0	4.9	89.54 ± 9.3	0.0
	1000	114.97	$^{**}\mathbf{126.37} \pm 4.0$	0.0	126.85 ± 3.9	0.0	127.08 ± 4.0	0.0	172.61 ± 7.1	0.0
r, 2500	100	23.47	25.99 ± 21.3	**57.3**	26.07 ± 21.5	57.0	$^{*}\mathbf{25.95} \pm 21.1$	57.0	35.42 ± 23.1	1.6
	500	115.24	$\mathbf{133.94} \pm 21.3$	4.7	134.25 ± 21.5	**5.9**	133.99 ± 21.2	4.1	177.25 ± 21.2	0.0
	1000	229.95	$\mathbf{269.99} \pm 21.6$	0.1	270.17 ± 21.7	**0.4**	270.03 ± 21.6	0.1	355.04 ± 21.2	0.0
r, 5000	100	11.98	$\mathbf{17.51} \pm 47.5$	$^{*}\mathbf{48.0}$	17.54 ± 47.3	46.8	17.54 ± 47.2	46.2	29.61 ± 32.7	0.5
	500	57.87	$^{**}\mathbf{93.59} \pm 43.0$	**8.7**	94.18 ± 43.0	8.6	93.97 ± 42.9	8.0	153.42 ± 28.9	0.0
	1000	115.23	$^{**}\mathbf{192.38} \pm 42.7$	**3.1**	192.92 ± 42.8	2.6	192.79 ± 42.7	3.0	308.64 ± 28.7	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 the theoretical minimum of t bins.

It is immediately evident that all EA versions outperform the MFFD$^+$ heuristic, with superior averages in all cases. As $|\mathcal{I}|$ and W are increased, the local search procedure takes longer as the rise in both the number of bins in solutions and the number of items per bins results in many more applications of AHC; fewer iterations of the EA are performed. For example, the average number of iterations ranged from 360,000 to 1200 when $W = 2500$ for artificial instances using 100 and 1000 items respectively, whilst the corresponding figures for real instances using $W = 5000$ were 55,000 and 65 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 size. The reduction in the number of EA iterations means that the evolution of the population is restricted which prevents better solutions from being achieved, suggesting that the algorithm is less accurate.

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

The different recombination operators also seem to perform well with different types of problem instance. GGA consistently produces the best results for both artificial and real instances using the larger bin size, where high quality solutions contain approximately 8.6 items per bin on average. Here, it seems that the lack of bias in choice of bins to inherit allows more of the solution space to be explored, resulting in better solutions

being found. AGX′ also appears to perform well when $W = 2500$, where the best solutions average 4.3 items per bin. With regards to the average number of bins $|\mathcal{S}|$, AGX′ produces the best solutions for three of the six instance classes using the smaller bin size. For the remaining three classes, the result from AGX′ uses at most 0.05 bins more than the best result found. By selecting bins containing the most items, AGX′ aims to build solutions comprising fewer yet fuller bins, thus AGX′ is focused on higher quality individual bins as opposed to entire solutions.

On the whole, AGX generates the least favourable results, particularly for real instances where it is outperformed by GGA and AGX′ in all six instance classes with regards to the average number of bins per solution $|\mathcal{S}|$. Selecting bins based on fullness may mean choosing bins containing fewer items, thus requiring extra bins to pack all items. However, note that AGX yields the highest number of solutions comprising t bins in three of the instance classes, suggesting a high variance in the quality of solutions produced.

It was also observed that instances in classes where solutions comprise a smaller number of items per bin on average (i.e. when $W = 2500$) converged early in runs, whereas classes with a higher number of items per bin take much longer to converge; some may even benefit from extended run times.

5. Conclusion and Further Work

This paper addressed the Score-Constrained Packing Problem (SCPP), a one-dimensional packing problem which involves packing items into a minimal number of bins such that the order and orientation of items within each bin satisfies the vicinal sum constraint. We began by describing the Alternating Hamiltonian Construction (AHC) algorithm for the sub-SCPP, an exact polynomial-time algorithm that can identify a feasible packing of items in a single bin. We then presented an evolutionary algorithm (EA) framework for the SCPP comprising a local search procedure and three distinct recombination operators. Experimental analysis comparing the recombination operators showed that selecting bins randomly (GGA) and selecting bins containing a higher number of items (AGX′) is preferable over simply chooding bins based on the overall capacity (AGX), as GGA encourages a wider search of the solution space, whilst AGX′ prioritises bins where the vicinal sum constraint is fulfilled between as many items as possible. Although AGX is an obvious choice for the classical BPP, its poor results show the importance of the vicinal sum constraint, which is a focal point of the SCPP.

One possible extension of the work in this paper is to use the EA to produce a pool of high quality solutions or bins, which could then be used to solve the Minimum Cardinality Exact Cover Problem (see Hawa (2020)) and find a superior solution using a commercial solver. Another beneficial avenue involves seeking a more accurate lower bound t for the SCPP which takes into account the proportion of score widths that satisfy the vicinal sum constraint with respect to the minimum scoring distance τ. Finally, it would be interesting to study the effects of combining or alternating the use of the GGA and AGX′ recombination operators within the EA, and identifying which operator should be used in each iteration depending on the parent solutions selected.

References

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

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.

Becker, K. H., & Appa, G. (2015). A Heuristic for the Minimum Score Separation Problem, a Combinatorial Problem Associated with the Cutting Stock Problem. *Journal of the Operational Research Society*, *66*, 1297–1311.

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

Bennell, J. A., Cabo, M., & Martinez-Sykora, A. (2018). A beam search approach to solve the convex irregular bin packing problem with guillotine cuts. *European Journal of Operational Research*, *270*, 89–102.

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

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

Coelho, K. R., Cherri, A. C., Baptista, E. C., Jabbour, C. J. C., & 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., Garey, M. R., & Johnson, D. S. (1984). Approximation Algorithms for Bin-Packing – An Updated Survey. In *Algorithm Design for Computer System Design* (pp. 49–106). Springer.

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

Cui, Y., Song, X., Chen, Y., & 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*, 269–280.

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

Dósa, G. (2007). The tight bound of first fit decreasing bin-packing algorithm is $FFD(I) \leq 11/9OPT(I) + 6/9$. In *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies* (pp. 1–11). Springer.

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

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*, 79–102.

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

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

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

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

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

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

Garey, M. R., & 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., & Wauters, T. (2016). The one-dimensional cutting stock problem with sequence-dependent cut losses. *International Transactions in Operational Research*, *23*, 5–24.

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

Gilmore, P. C., & Gomory, R. E. (1963). A linear programming approach to the cutting stock problem—part ii. *Operations research*, *11*, 863–888.

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

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

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

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

Hawa, A. L. (2019a). Evolutionary Algorithm (EA) source code and results for the article "Exact and Approximate Methods for the Score-Constrained Packing Problem". URL: https://doi.org/10.5281/zenodo.3374418. doi:10.5281/zenodo.3374418.

Hawa, A. L. (2019b). Problem instance generator for the Score-Constrained Packing Problem. URL: https://doi.org/10.5281/zenodo.2599621. doi:10.5281/zenodo.2599621.

Hawa, A. L. (2020). *Exact and Evolutionary Algorithms for the Score-Constrained Packing Problem*. PhD Thesis Cardiff University.

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

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

He, K., Tole, K., Ni, F., Yuan, Y., & Liao, L. (2021). Adaptive large neighborhood search for solving the circle bin packing problem. *Computers & Operations Research*, *127*, 105140.

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

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

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

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

Levine, J., & Ducatelle, F. (2004). Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, *55*, 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*, 2295–2310.

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

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

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

Liu, Q., Cheng, H., Tian, T., Wang, Y., Leng, J., Zhao, R., Zhang, H., & Wei, L. (2021). Algorithms for the variable-sized bin packing problem with time windows. *Computers & Industrial Engineering*, *155*, 107175.

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

Ma, N., Liu, Y., & 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., & Peled, U. N. (1994). Longest cycles in threshold graphs. *Discrete Mathematics*, *135*, 169–176.

Malaguti, E., Monaci, M., & Toth, P. (2008). A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, *20*, 302–316.

Martello, S., & Toth, P. (1990). Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, *28*, 59–70.

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

Ntene, N., & van Vuuren, J. H. (2008). A survey and comparison of heuristics for the 2D oriented on-line strip packing problem. *ORiON*, *24*, 157–183.

Poldi, K. C., & Arenales, M. N. (2009). Heuristics for the one-dimensional cutting stock problem with limited multiple stock lengths. *Computers & operations research*, *36*, 2074–2081.

Quiroz-Castellanos, M., Cruz-Reyes, L., Torres-Jimenez, J., Gómez, C., Huacuja, H. J. F., & 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* (pp. 222–229). volume 91.

Rekiek, B., De Lit, P., Pellichero, F., Falkenauer, E., & Delchambre, A. (1999). Applying the equal piles problem to balance assembly lines. In *International Symposium on Assembly and Task Planning* (pp. 399–404). IEEE.

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

Thompson, J. M., & Dowsland, K. A. (1998). A robust simulated annealing based examination timetabling system. *Computers & Operations Research*, *25*, 637–648.

Wäscher, G., Haußner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, *183*, 1109–1130.

Wu, D., & 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*, 2182–2189.

Xavier, E., & Miyazawa, F. K. (2008). A one-dimensional bin packing problem with shelf divisions. *Discrete Applied Mathematics*, *156*, 1083–1096.

Yuan, Y., Tole, K., Ni, F., He, K., Xiong, Z., & Liu, J. (2021). Adaptive simulated annealing with greedy search for the circle bin packing problem. *arXiv preprint arXiv:2108.03203*, .

Zhou, C., Wu, C., & Feng, Y. (2009). An exact algorithm for the type-constrained and variable sized bin packing problem. *Annals of Operations Research*, *172*, 193.