An investigation into two bin packing problems with ordering and orientation implications

R. Lewis[1], X. Song[1], K. Dowsland[2] and J. Thompson[1]
[1]Cardiff School of Mathematics,
Prifysgol Caerdydd/Cardiff University, Cardiff CF24 4AG, WALES.
email: lewisR9|songX6|thompsonJM1@cf.ac.uk
[2]Gower Optimal Algorithms Ltd,
5 Whitestone Lane, Newton, Swansea SA3 4UH, WALES.
email: k.a.dowsland@btconnect.com

September 30, 2011

Abstract

This paper considers variants of the one-dimensional bin packing (and stock cutting) problem in which both the ordering and orientation of items in a container influences the validity and quality of a solution. Two new real-world problems of this type are introduced, the first that involves the creation of wooden trapezoidal-shaped trusses for use in the roofing industry, the second that requires the cutting and scoring of rectangular pieces of cardboard in the construction of boxes. To tackle these problems, two variants of a local search-based approximation algorithm are proposed, the first that attempts to determine item ordering and orientation via simple heuristics, the second that employs more accurate but costly branch-and-bound procedures. We investigate the inevitable trade-off between speed and accuracy that occurs with these variants and highlight the circumstances under which each scheme is advantageous.

1 Introduction

Many problems arising in mathematics, computer science and operational research require the partitioning of a discrete set of entities into an exhaustive collection of mutually exclusive subsets, subject to specific constraints. Though simple to define, such "grouping" problems often pose significant challenges in practical settings because it is not always easy to judge whether the imposed constraints can be satisfied. Indeed many such problems including those concerning graph partitioning (Hertz et al., 2008; Isomoto et al., 1993; Jensen and Toft, 1994; Nakano1 et al., 1995), school and university timetabling (Lewis, 2008; McCollum et al., 2010), sports fixture scheduling (de Werra, 1988; Kendall et al., 2010; Rasmussen and Trick, 2008), load balancing (Falkenauer, 1998), and frequency assignment (Aardel et al., 2002; Valenzuela, 2001), are known to be NP-hard (Garey and Johnson, 1979; Karp, 1972), implying that we cannot hope to establish polynomially bounded algorithms for solving them in the general sense.

An important class of grouping problem common in various commercial operations such as the clothing and construction industries are *packing* and *cutting* problems. Such problems involve determining the way in which a pre-defined set of "items" should be "packed" into "bins" (respectively "cut" from "stocks") such that wastage (or spare capacity) is minimised. In this paper we focus on a subclass of these, what Wäscher et al. (2007) have termed *fixed-dimension input-minimisation*

problems: given a set of items $\mathcal{I}$ (where $|\mathcal{I}| = n$), construct a set of "groups" $\mathcal{S} = \{S_1, S_2, \ldots, S_{|\mathcal{S}|}\}$, where:

$$\bigcup S_i = \mathcal{I} \tag{1}$$

$$S_i \cap S_j = \emptyset \quad (\text{for } 1 \leq i < j \leq |\mathcal{S}|) \tag{2}$$

$$S_i \in \mathcal{F} \quad (\text{for } 1 \leq i \leq |\mathcal{S}|), \tag{3}$$

such that the number of groups $|\mathcal{S}|$ is minimised.

In the above, each group $S_i \in \mathcal{S}$ defines the items that will be packed into the same bin (respectively, cut from the same stock), and conditions (1) and (2) specify that each item should be assigned to exactly one of the $|\mathcal{S}|$ groups. Condition (3), meanwhile, indicates that each of these groups should be *feasible*: thus the set $\mathcal{F}$ denotes the set of all feasible subsets of $\mathcal{I}$. The contents of $\mathcal{F}$ are affected by the constraints of the specific problem being considered, and will also vary on an instance-by-instance basis. For non-trivial instances, $\mathcal{F}$ will also usually be too large to be constructed directly since the number of possible groups (subsets) of items to be checked rises exponentially with respect to n.

Perhaps the most widely known examples in this class of problem are the one-dimensional stock cutting problem (SCP) and the one-dimensional bin packing problem (BPP). In both cases each item $j \in \mathcal{I}$ has a size s_j, and the aim is to partition the items into groups such that $\forall S_i \in \mathcal{S}$, $\sum_{j \in S_i} s_j \leq C$, where C is some constant defined as part of the problem. According to the taxonomy of Wäscher et al. (2007) the main distinction between these problems is that in the BPP the items are strongly heterogeneous (i.e. of many different sizes), whereas in the SCP items are only weakly heterogeneous (that is, many items have the same size). Though rather subjective, this difference is important to note as, historically, it has brought about quite different solution approaches for the two problems (Falkenauer, 1998; Haessler and Sweeny, 1991; Liang et al., 2002; Martello and Toth, 1990b; Lewis, 2009).

Although the SCP and BPP are both NP-hard (Garey and Johnson, 1979), it is noticeable that the sub-problem of determining whether $S_i \in \mathcal{F}$ in these cases is trivial, since we need only sum the sizes of the items assigned to S_i. The commutativity of this operation also implies that the *order* in which items occur in a bin/on a stock is irrelevant since all of the $|S_i|!$ permutations of items in a group will result in the same total. In many practical circumstances however, things can be more complicated and factors beyond the total size of the items might also need to be considered for determining membership of $\mathcal{F}$. In this paper we will examine two such problems where both the ordering and orientation of items is significant in the sub-problem of deciding whether $S_i \in \mathcal{F}$. We will observe that in both cases these underlying sub-problems are NP-complete, adding an extra degree of complexity to the grouping problem. Having noted these features we propose a local search-based method for these problems and examine the effects of tackling the underlying sub-problems using fast but approximate heuristics, and also using more accurate but expensive branch-and-bound procedures.

The problems considered in this paper involve the creation of wooden trusses for use in the roofing industry, and the cutting and scoring of rectangular pieces of cardboard for use in the construction of packing boxes. In the next section we begin by describing the overall algorithmic framework used for both of these problems. In Sections 3 and 4, we then conduct an analysis of each problem in turn and describe the operators that are used in conjunction with this framework, presenting empirical evidence on the subsequent performances. In general, we tend to refer these problems as bin packing variants, since this seems the most appropriate classification according to the taxonomy of Wäscher et al. (2007); thus we will often speak of items being "packed" into "bins" (though in the real industrial processes, what is actually required is for items to be cut from

stocks). Finally, Section 5 concludes the paper and makes some suggestions for further work on these new problems.

2 Algorithm Framework

The algorithmic framework we propose for both problems is based on a stochastic method that was originally developed by Lewis (2009) for the BPP. We specifically choose this method as its execution times are very fast, but yet it is still able to produce solutions of equal quality to those achieved by other well-known packing algorithms such as the hybrid grouping genetic algorithm of Falkenauer (1998) and the ant-based algorithm of Levine and Ducatelle (2003). The former attribute is particularly useful because of the extra time that is consumed in determining groups' membership of $\mathcal{F}$, a task that inevitably has to be performed many times during a run.

The basic method starts by producing a feasible solution $\mathcal{S}$ (that is, a solution satisfying conditions (1), (2), and (3) above) but where $|\mathcal{S}|$ is not limited. Further details on how this can be achieved for each of our problems are given in Sections 3 and 4. During a run, attempts are then made via two search operators to try and reduce $|\mathcal{S}|$ by shuffling items between groups, ensuring that all groups in the solution remain feasible at all times. A single iteration of the algorithm operates as follows. Given a solution $\mathcal{S}$, e.g.

$$\mathcal{S} = \{\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}, \{\mathsf{d}, \mathsf{e}, \mathsf{f}\}, \{\mathsf{g}, \mathsf{h}\}, \{\mathsf{i}\}, \{\mathsf{j}, \mathsf{k}\}\}$$

a small number of groups in $\mathcal{S}$ are first randomly selected and moved into a second set $\mathcal{T}$, e.g.

$$\mathcal{S} = \{\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}, \{\mathsf{g}, \mathsf{h}\}, \{\mathsf{i}\}\}$$
$$\mathcal{T} = \{\{\mathsf{d}, \mathsf{e}, \mathsf{f}\}, \{\mathsf{j}, \mathsf{k}\}\}$$

The Local-Search procedure (fig. 1) is then applied to $\mathcal{S}$ and $\mathcal{T}$. This is a modified version of a procedure previously used with the BPP (Falkenauer, 1998; Levine and Ducatelle, 2003; Lewis, 2009) and is based on the concepts of *dominance*, defined by Martello and Toth (1990b). The idea is that items are interchanged between groups in $\mathcal{S}$ and groups in $\mathcal{T}$ such that the number of items in each group in $\mathcal{S}$ remains the same or decreases, while the total size of the items within these groups increases. If this is achieved, then the groups in $\mathcal{S}$ can be said to have improved since the amount of wastage in these groups will have decreased. Also the smaller items of the problem (which could be easier to deal with as they might be used to "fill in gaps") are moved into groups in $\mathcal{T}$. As is described in the pseudocode in fig. 1, the Local-Search procedure operates by first attempting to (feasibly) swap a pair of items from some group in $\mathcal{S}$ with a pair of items from a group in $\mathcal{T}$ (lines 2-7); next, attempts are made to swap a pair of items from a group in $\mathcal{S}$ with a single item from a group in $\mathcal{T}$ (lines 8-13); finally single items from a group in $\mathcal{S}$ are swapped with single items from a group in $\mathcal{T}$ (lines 14-19). This process is continued until an entire parse is performed with no changes being made to any of the groups: thus the solution can be considered at a local optima from the perspective of this search operator.

On completion of Local-Search, we will still have two sets $\mathcal{S}$ and $\mathcal{T}$, though their contents are likely to have changed, e.g.

$$\mathcal{S} = \{\{\mathsf{a}, \mathsf{e}\}, \{\mathsf{g}, \mathsf{k}\}, \{\mathsf{i}\}\}$$
$$\mathcal{T} = \{\{\mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{f}\}, \{\mathsf{h}, \mathsf{j}\}\}$$

```
LOCAL-SEARCH (S = {S_1, ..., S_|S|}, T = {T_1, ..., T_|T|})
(1)  for g ← 1 to |S|
(2)      foreach pair of items {i, j} ∈ S_g
(3)          for h ← 1 to |T|
(4)              foreach pair of items {k, l} ∈ T_h
(5)                  δ ← (A(k) + A(l)) − (A(i) + A(j))
(6)                  if δ > 0 and ((S_g ∪ {k, l}) \ {i, j}) ∈ F and ((T_h ∪ {i, j}) \ {k, l}) ∈ F
(7)                      Move items i, j from S_g to T_h and move items k, l from T_h to S_g
(8)      foreach pair of items {i, j} ∈ S_g
(9)          for h ← 1 to |T|
(10)             foreach item k ∈ T_h
(11)                 δ ← A(k) − (A(i) + A(j))
(12)                 if δ > 0 and ((S_g ∪ {k}) \ {i, j}) ∈ F and ((T_h ∪ {i, j}) \ {k}) ∈ F
(13)                     Move items i, j from S_g to T_h and move item k from T_h to S_g
(14)     foreach item i ∈ S_g
(15)         for h ← 1 to |T|
(16)             foreach item k ∈ T_h
(17)                 δ ← A(k) − A(i)
(18)                 if δ > 0 and ((S_g ∪ {k}) \ {i}) ∈ F and ((T_h ∪ {i}) \ {k}) ∈ F
(19)                     Move item i from S_g to T_h and move item k from T_h to S_g
```

```
FIRST-FIT (S = ∅, π)
(1)  for i ← 1 to |π|
(2)      found ← false
(3)      for j ← 1 to |S|
(4)          if S_j ∪ {π_i} ∈ F
(5)              S_j ← S_j ∪ {π_i} /*Item inserted into an existing group*/
(6)              found ← true
(7)              break
(8)          else j ← j + 1
(9)      if not found
(10)         S_j ← {π_i} /*Create a new group for the item*/
(11)         S ← S ∪ {S_j} /*Add this new group to the solution*/
```

Figure 1: The two main procedures used in the presented algorithms. Here the notation $A(i)$ refers to the area of an item i, and $\mathcal{F}$ denotes the set of all feasible groups (thus $\forall S_i \in \mathcal{S}$, $S_i \in \mathcal{F}$ and $\forall T_i \in \mathcal{T}$, $T_i \in \mathcal{F}$). Note also that when First-Fit is called, the permutation π will contain all items, and on completion $\mathcal{S}$ will be a complete and feasible solution.

At this point the groups of both $\mathcal{S}$ and $\mathcal{T}$ are used to produce a permutation of the n items $\pi = (\pi_1, \ldots, \pi_n)$. This permutation is formed such that (a) items currently within the same group are placed into adjacent positions in π, and (b) according to some (possibly random) ordering of the groups. For example, ordering the groups by decreasing cardinality might lead to the permutation:

$\pi =$(b, c, d, f, a, e, h, j, g, k, i).

Finally, a new solution is constructed by feeding π into the First-Fit algorithm (fig. 1), completing a single iteration of the algorithm. First-Fit is a classical method for grouping problems that operates by taking each item π_j in turn ($1 \leq j \leq n$), placing it into the lowest indexed group for which it is feasible, and creating new groups where necessary. First-Fit is useful here because its execution often results in a solution that is quite different to the solution used to form π, thus it can help the algorithm to escape the local optima achieved by Local-Search, bringing the possibility of further improvements being made in a subsequent iteration. A second property that also occurs for some types of problem is that First-Fit is guaranteed to produce a solution that

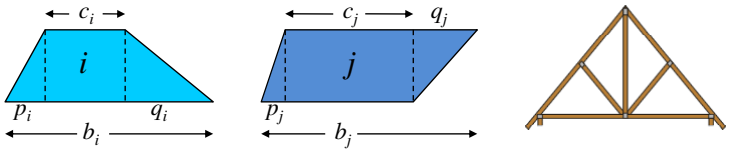


Figure 2: (Left, middle): Example items with projections on the same side, and on alternate sides. (Right): A simple roof truss formation.

has equal or fewer groups than its predecessor. The specific criterion that needs satisfying for this to be true is:

$$S_i \in \mathcal{F} \Rightarrow \forall s \subset S_i,\ s \in \mathcal{F}. \tag{4}$$

This condition is known to exist for many grouping problems including the BPP, SCP, and graph colouring problem (Culberson and Luo, 1996; Lewis, 2009). It is also satisfied for the Truss Cutting Problem considered in Section 3, though this cannot be said of the Box Cutting Problem, as we discuss further in Section 4.

3 Problem A: The Truss Cutting Problem (TCP)

The first problem considered in this study arises in the roofing industry and was originally brought to our attention due to an enquiry made to the software company of this paper's third author. The problem involves cutting large numbers of wooden trusses that are to be used in roof construction. These trusses are of different sizes and are to be cut from wooden boards in such a way that the number of boards used is minimised. Though similar to the classical BPP, this problem is complicated by the fact that the trusses are trapezoidal in shape with their ends involving different angles to be cut in either the "/" or "\" direction. This means that in addition to having to decide which trusses to cut from which board, solution quality is also affected by the ordering and orientations of the trusses.

The Truss Cutting Problem (TCP) can be more formally stated as follows: We are given a set of rectangular bins of height H and width W, and a set $\mathcal{I}$ of n items (trusses). Each item $j \in \mathcal{I}$ is defined as a trapezoid with height H and three further values: a "base width" b_j, and two "projections" p_j and q_j that define the internal angles of each item. We assume that $\max(b_j | j \in \mathcal{I}) \leq W$. It is also convenient to define a "central width" $c_j = b_j - (p_j + q_j)$ for each item and, in addition, the projections on an item can occur on the same side or alternate sides of the trapezoid (see fig. 2). In all cases an item's area $A(j)$ is simply:

$$A(j) = \frac{1}{2} H(b_j + c_j). \tag{5}$$

Figure 3 demonstrates the way in which the arrangement of such items in a bin can determine the amount of *inter-item wastage* incurred (in our case we define "inter-item wastage" as the total area of all triangular spaces between each adjacent pair of projections, plus the left- and right-most triangles of waste, as shown in the figure). We see that the bottom example in the figure features less inter-item wastage and, as a result, has a larger amount of contiguous unoccupied area on the right side of the bin which might be used to contain further items.

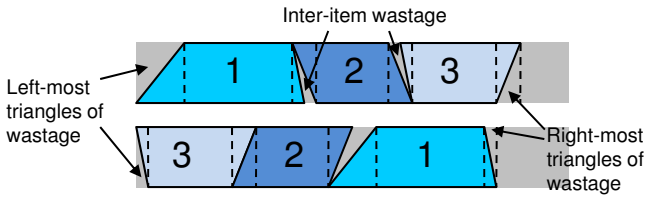


Figure 3: Example of how items can be arranged in a bin to minimise inter-item wastage.

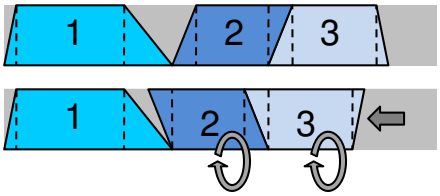


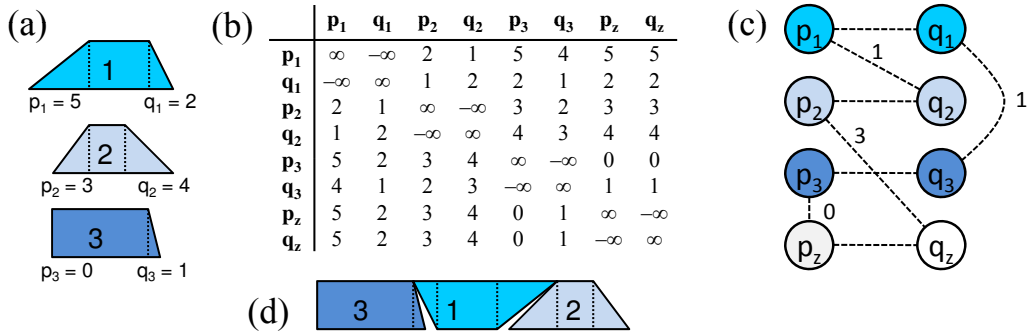
Figure 4: Illustrating how joins between items can be nested by flipping them on their horizontal axes.

We also observe that items can be placed into a bin according to four orientations: they can be "flipped" on none, either, or both of their vertical and horizontal axes. However, in terms of minimising inter-item wastage we actually only need to consider two of these orientations – specifically, we only need to decide whether each item should be flipped on its vertical axis. This is proved, via contradiction, as follows:

Suppose $|S_i|$ items are positioned in a bin in a given order and with specified orientations on the vertical axes. If the orientations of the items on their horizontal axes are such that the inter-item wastage of this configuration is minimised, then it is clear that adjacent items will be aligned so that they "nest", as is the case with the item configurations in fig. 3. However, suppose that the opposite is true and two adjacent pieces do not nest, as is the case with items 1 and 2 in the top example of fig. 4. If we now take all items to the right of this join and flip them on their horizontal axis, then observe that this join will now be nested and that inter-item wastage will have decreased, with the rest of the arrangement remaining unchanged. Thus, the original orientation of the items could not have been optimal.

The above observation implies that the task of arranging items in a bin thus involves (a) determining the order of the items from left to right, and (b) deciding for each item j, whether projection p_j or q_j should occur on the left. Adjacent items can then be nested (i.e. orientation on the horizontal axes can be determined) according to this information. This also implies that the wastage between any two projections p_j and p_k (w.l.o.g) is always $\frac{1}{2}H(|p_j - p_k|)$.

If we are to determine an effective method for the TCP, it is thus obvious that in addition to requiring operators that decide which items to assign to which bin (such as the local search method of Section 2), we also need to determine high-quality arrangements of these items within each bin. Under our algorithmic operators, this is characterised by the task of determining whether a group of items S_i is a member of $\mathcal{F}$. This task is related to what we call the Truss Sub-problem, which we now examine in more detail.



	p_1	q_1	p_2	q_2	p_3	q_3	p_z	q_z
p_1	∞	$-\infty$	2	1	5	4	5	5
q_1	$-\infty$	∞	1	2	2	1	2	2
p_2	2	1	∞	$-\infty$	3	2	3	3
q_2	1	2	$-\infty$	∞	4	3	4	4
p_3	5	2	3	4	∞	$-\infty$	0	0
q_3	4	1	2	3	$-\infty$	∞	1	1
p_z	5	2	3	4	0	1	∞	$-\infty$
q_z	5	2	3	4	0	1	$-\infty$	∞

Figure 5: Example conversion of a truss sub-problem to a TSP. A valid route (involving all $-\infty$ arcs) represents a valid formation of the items. Here $H = 2$.

3.1 Examination of the Truss Sub-problem

Consider a problem where we have a group of truss items $S_i \subseteq \mathcal{I}$ and we wish to determine the order and orientation of all items in S_i such that inter-item wastage is minimised. We now observe that this problem can be naturally reduced to an instance of the NP-hard symmetric travelling salesman problem (TSP). We perform this conversion by considering each of the $2|S_i|$ projections as a city, with the distances between each pair of cities being defined as the wastage between the two associated projections. For projections on the same item, the inter-city distance is defined as $-\infty$, and it is also necessary to introduce two "dummy cities" p_z and q_z, which are used to represent the left- and right-most triangles of wastage in a bin. The distances between dummy cities and any other city p_j (resp. q_j) is simply $\frac{1}{2}Hp_j$ (resp. $\frac{1}{2}Hq_j$) and the arc between the pair of dummy cities is also set to $-\infty$. Any Hamiltonian cycle that visits all cities exactly once and that traverses all $-\infty$ arcs is considered a "valid" route, and an *optimal* valid route for this model thus corresponds to a configuration of items featuring minimal inter-item wastage.

An example distance matrix D for $|S_i| = 3$ items (and thus 8 cities) is given in fig. 5(b). Fig. 5(c) then shows an example of a valid route, with the labelled arcs indicating the inter-item wastage incurred in the corresponding arrangement of items, shown in fig. 5(d). Specifically, there is one unit of wastage between item 3 and 1, one unit between 1 and 2, and zero and 3 units of wastage in the left-most and right-most triangles respectively. The total number of valid routes (and thus distinct item arrangements) for $|S_i|$ items is $(2^{|S_i|}|S_i|!)/2$ which, for this example, is 24.

Definition 1.

The truss sub-problem asks: Given a bin of dimensions $H \times W$, and given $S_i \subseteq \mathcal{I}$ such that $A(S_i) = \sum_{j \in S_i} A(j) \leq HW$, is $S_i \in \mathcal{F}$?

To answer the truss sub-problem we are interested in determining the existence of an item configuration whose inter-item wastage is less than or equal to $(HW - A(S_i))$.[1] Of course, this is equivalent to the NP-complete decision variant of the corresponding TSP, where we are asked to determine a valid route whose total distance (discounting $-\infty$ arcs) is less than $(HW - A(S_i))$.

Having noted the underlying intractability of the truss sub-problem, we now note two special cases which are, in fact, polynomially solvable.

[1] Recall that if the inter-item wastage plus $A(S_i)$ is less than HW, then there will also be unoccupied area on the right side of the bin, as is the case in fig. 3.

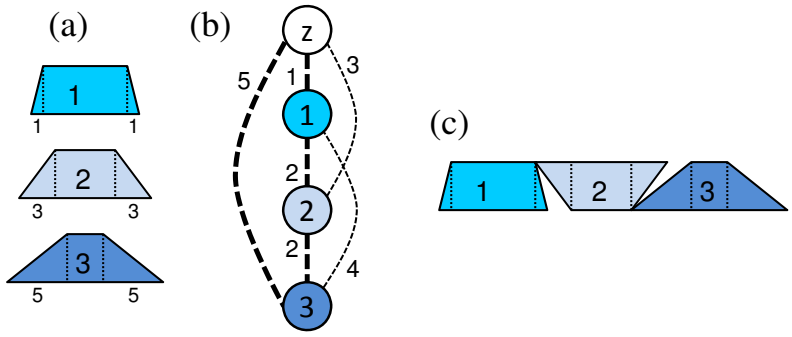


Figure 6: Resultant graph when all items are symmetric about their vertical axis. The bold lines in (b) represent an optimal route and (c) shows the corresponding arrangement of the items.

3.1.1 Case 1: All projections are of size zero

In this case all items $j \in S_i$ will be rectangular in shape and so the wastage between any two items (and in the left- and right-most triangles) will be zero. In the corresponding TSP model this means that all non-$(-\infty)$ arcs have a distance of zero, implying that any valid route will be optimal (and thus all arrangements of items in a bin are optimal). More generally, if the n items of the overall TCP feature projections of zero, then the problem becomes equivalent to the one-dimensional BPP and SCP.

3.1.2 Case 2: All items are Isosceles trapezoids and parallelograms

The second special case arises when $p_j = q_j, \forall j \in S_i$ (that is all items in a group S_i are symmetric about their vertical axes). In the corresponding TSP the destinations and distances of arcs emanating from each pair of cities (p_j, q_j) will be identical; thus we can merge each pair into a single city (see fig. 6). Now, if the $|S_i|$ items are sorted according to the sizes of their projections, the resultant graph will be a special case of Euclidean graph in which all cities lie on a straight line.[2] As demonstrated in fig. 6, for such graphs optimal routes can then be obtained by visiting each city in the sorted order (in this case from top to bottom) and then returning to the initial (dummy) city. From the perspective of arranging items, the equivalent strategy involves placing each item into the bin in ascending order of projection sizes.

3.2 Addressing the Truss Sub-Problem

According to our algorithmic framework (Section 2) instances of the truss sub-problem will be faced many times during a run, but for larger values of $|S_i|$ there may be instances that cannot be solved in reasonable time. Thus an appropriate balance needs to be struck between efficiency and accuracy. In our case, on being presented with a group of items S_i (such that $A(S_i) \leq HW$) the following tests are first carried out. If one of these tests solves the sub-problem, remaining tests are not necessary.

- If $|S_i| = 1$ then $S_i \in \mathcal{F}$ (since we have assumed that $\max(b_j | j \in \mathcal{I}) \leq W$).

[2] That is, if the distances between three successive cities a, b, and c are $D_{a,b}$ and $D_{b,c}$ respectively, then $D_{a,c} = D_{a,b} + D_{b,c}$.

- If $A(S_i) = HW$, then in order for $S_i \in \mathcal{F}$ to be true, an arrangement must exist in which no wastage exists around any of the items. Thus for each projection on each item, there must exist an equi-sized projection on some other item in S_i. It is also necessary that there must exist at least two projections on distinct items that are of size zero as otherwise we would be guaranteed to incur wastage on one or both of the left- and right-most triangles in the bin. If any of these conditions are not met then $S_i \notin \mathcal{F}$.

- If $HW - A(S_i) < \tau_{\max}$ then $S_i \notin \mathcal{F}$. Considering the TSP model described above, the value $\tau_{\max}$ is calculated by considering each pair of cities (p_j, q_j) and observing the value τ_j which is the total of the shortest non-$(-\infty)$ arcs emanating from city p_j and city q_j respectively. Each value τ_j thus represents the minimum amount of wastage that we can hope to exist on the left- and right-hand side of a particular item j. Of course, it is impossible to pack all items into a bin if $\tau_{\max} = \max(\tau_j | j \in S_i) > HW - A(S_i)$.

In our experience we have found that approximately half of the truss sub-problems encountered during runs with our test problems are solved by these tests, though this proportion can alter for different sorts of problem (see Section 3.5). When these tests are insufficient, our next step is to try and fit the items into a bin. In our case this is carried out using a simple greedy method based on the nearest neighbour strategy for the TSP: i.e. Starting at dummy city p_z, at each stage move to the nearest unvisited city, and once all cities have been visited, return to the first city p_z. This method is exact for the two special cases mentioned in Section 3.1, though in the general case this is not so. Thus, if upon completion of this step the resultant inter-item wastage is larger than $HW - A(S_i)$, a second greedy process is invoked that attempts to improve the arrangement by reversing the orientation of some of the items. Specifically, the process considers each item in a bin in turn from left to right and identifies whether rotating this item on its vertical axis will reduce the total amount of inter-item wastage. If this is the case then the rotation of the item is performed.

In the first version of our algorithm (Algorithm A), if the greedy processes above have resulted in an item configuration whose inter-item wastage is larger than $HW - A(S_i)$, then it is assumed that $S_i \notin \mathcal{F}$. However, it is possible that in many cases the opposite is true – thus in version B of the algorithm more powerful methods are employed at this point. We achieve this using an integer programming formulation for the TSP (given in Appendix A). Note that we do not necessarily need to solve each sub-problem to optimality as we only need to know if there is a solution whose objective function (eq. (8)) is less or equal to $HW - A(S_i)$. Thus, if we use a branch-and-bound approach, we can stop the search when:

- the global lower bound exceeds $HW - A(S_i)$ (proving $S_i \notin F$);

- the global upper bound does not exceed $HW - A(S_i)$ (proving $S \in F$);

- a predefined CPU time limit for the process is reached.

If the final criterion above is met then it is assumed that $S_i \notin \mathcal{F}$. Of course, this may not be the case, but the assumption is necessary to avoid excessive run times, which could occur with larger values of $|S_i|$. In our case we found a time limit of two seconds to be sufficient in this regard. We implement the above strategy with the optimisation software Xpress which uses the standard branch-and-bound algorithm for integer programming based on linear programming relaxations. Thus the upper bound used in the second criterion above corresponds to the best integer solution to date and provides a valid arrangement of the items in a bin.

Due to the various overheads associated with calling and running Xpress, it makes sense to invoke the B&B process only when necessary, which in our case is when the cheaper greedy methods

are unable to fit the items into a bin. In addition, we also find that performance can be considerably improved by disallowing B&B to be used more than once with any particular subset of items. This can be achieved by maintaining two sets $\mathcal{X}$ and $\mathcal{Y}$ during run time, the former that contains the subsets of items that B&B has determined *can* be packed into a single bin (together with the corresponding arrangement), and the second containing all subsets that B&B has found *cannot* be packed into a single bin. In our implementation, these sets are stored as binary trees, allowing elements to be added and searched-for in logarithmic time. Of course, during an extended run it is possible that the cardinality of these sets may become very large; however, in practice many groups of items considered during a run will have their membership of $\mathcal{F}$ confirmed or denied before invocation of B&B has become necessary and are thus not added to $\mathcal{X}$ or $\mathcal{Y}$. In addition, in many TCPs it is also common to see multiple occurrences of items with the same dimensions (that is, items of the same "type"). Thus with appropriate book-keeping, individual elements in $\mathcal{X}$ or $\mathcal{Y}$ can actually be made to represent many different subsets of items in reality.

3.3 Problem Generation

To test our algorithm an instance generator was implemented that attempts to simulate the sort of problems encountered in the roofing industry.[3] We used this to produce a total of 1300 problem instances with sizes ranging from $n = 100$ to 500. Within this set, two kinds of problem were constructed: *artificial* ("a") instances, where every item $j \in \mathcal{I}$ is composed of different dimensions, and the perhaps more *realistic* ("r") instances, where we see multiple occurrences of items with the same dimensions (i.e. items of the same type). In all cases we used a bin size (board length) of $W = 4200$mm which we have observed to be a standard industrial size.[4] The item widths b_j were then each set between 300mm to 3600mm and were either selected uniform randomly from this range, or so that a particular number of "large" items (2700-3600mm) and "small" items (300-1800mm) were included. Projection sizes were then selected for the items such that end angles occurred in the range $30° - 90°$ and it was assumed for all items $j \in \mathcal{I}$ that $p_j + q_j \leq b_j$. We also ensured that at least one item in each instance featured a projection of size zero (that is, angle of $90°$).

One feature that we noticed early in our experiments with these instances was that their solutions tended to feature fewer than three items per group on average. While this might actually be the case in industrial settings, it does imply that the individual sub-problems encountered during a run will be rather small, with the underlying TSPs tending to feature < 8 cities. In addition, we also saw that the items in our simulated instances usually featured central widths much longer than the total size of their two projections. Again, this seems to be the case in practical circumstances because roof trusses are usually quite long and thin; however, this also suggests that the effects induced by wastage occurring between projections could be proportionally small, making the overall TCP similar to the classical BPP. To counter these features and thus gain a more comprehensive understanding of our algorithm, we thus chose to modify our original set of instances to produce two further sets. This was achieved by simply taking each item and setting the central width to half and quarter of its original value. The main features of the "original", "half", and "quarter" instance sets are listed in Table 1. Note that the number of different item types equals n for the artificial "a" instances, but is around 20 in the "r" instances.

[3] A detailed specification of the generator together with the complete set of instances can be found online at `www.rhydlewis.eu/resources/trapBoxProbs.zip`

[4] Note that bin height H is irrelevant in the solution process used here – thus, for simplicity's sake we assume $H = 2$, meaning that the wastage between two projections p_j and p_k (w.l.o.g) is simply $\frac{1}{2}H(|p_j - p_k|) = |p_j - p_k|$.

Table 1: Summary of Test Instances Used

Type, n	Probs.	Item Types[a]	TMin[b]	Best[c]	Items per group[d]	TMin obtained[e]
orig., a, 100	20	100.00	44.65	46.35	2.166	0.000
orig., a, 200	20	200.00	90.70	94.75	2.117	0.000
orig., a, 300	20	300.00	134.70	138.95	2.162	0.000
orig., a, 400	20	400.00	177.75	182.60	2.195	0.000
orig., a, 500	20	500.00	222.55	228.00	2.195	0.000
orig., r, 100	240	20.19	38.66	42.15	2.581	0.171
orig., r, 200	240	20.31	75.50	81.88	2.655	0.058
orig., r, 300	240	20.05	115.10	125.24	2.602	0.017
orig., r, 400	240	20.65	154.44	168.17	2.577	0.004
orig., r, 500	240	19.60	193.03	210.75	2.595	0.000
half, a, 100	20	100.00	23.45	23.75	4.217	0.700
half, a, 200	20	200.00	47.45	47.85	4.184	0.600
half, a, 300	20	300.00	70.55	70.90	4.234	0.650
half, a, 400	20	400.00	92.95	93.60	4.276	0.350
half, a, 500	20	500.00	116.40	117.25	4.267	0.150
half, r, 100	240	20.19	20.52	20.77	5.076	0.758
half, r, 200	240	20.31	39.97	40.44	5.207	0.550
half, r, 300	240	20.05	60.80	61.70	5.128	0.333
half, r, 400	240	20.65	81.34	82.38	5.106	0.233
half, r, 500	240	19.60	101.75	103.16	5.133	0.154
quar., a, 100	20	100.00	13.00	13.10	7.647	0.900
quar., a, 200	20	200.00	25.80	26.05	7.690	0.750
quar., a, 300	20	300.00	38.40	38.60	7.785	0.800
quar., a, 400	20	400.00	50.60	50.75	7.886	0.850
quar., a, 500	20	500.00	63.30	63.60	7.872	0.700
quar., r, 100	240	20.19	11.48	11.55	9.070	0.925
quar., r, 200	240	20.31	22.14	22.30	9.360	0.846
quar., r, 300	240	20.05	33.62	33.88	9.271	0.742
quar., r, 400	240	20.65	44.80	45.07	9.280	0.738
quar., r, 500	240	19.60	56.05	56.51	9.289	0.554

[a]Averaged across all instances
[b]TMin= $\lceil(\sum_{j=1}^{n} A(j))/HW\rceil$, averaged across all instances
[c]Number of groups used in the best run from either algorithm on each instance, averaged across all instances
[d]Calculated as n/Best, averaged across all instances
[e]Proportion of instances where Best = TMin

3.4 Bounds for the TCP

It is also possible to specify bounds on the optimal number of groups χ needed for a particular TCP instance. The most obvious of these is the theoretical minimum TMin $= \lceil(\sum_{j=1}^{n} A(j))/HW\rceil$. However, one problem with TMin is its tendency to underestimate χ when considering instances involving large items, because in these cases there might be various groups that contain only one or two items, perhaps resulting in rather a lot of (necessary) wastage. We thus also consider an alternative method of lower bound generation which can be found by simplifying a TCP instance problem to a corresponding BPP, and then solving this to optimality.

Consider the following:

Definition 2. ***Simplified Width****. For each item $j \in \mathcal{I}$ with base width b_j and projections p_j and q_j, define the* simplified width $s_j = b_j - \max(p_j, q_j)$.

Definition 3. ***Simplified Problem****. Assume all items $j \in \mathcal{I}$ from the original TCP problem are rectangular items with width s_j and height H. The simplified problem is now a BPP applied to these items using bin size $W \times H$.*

Theorem 1. *Let χ be the optimal (smallest) number of groups needed in a particular TCP instance, and let χ^s be the optimal in the corresponding simplified problem. Then $\chi^s \leq \chi$.*

A proof of Theorem 1 can be found in Appendix B. Of course, one issue with this method of bound generation is that the corresponding BPP, itself an NP-hard problem, needs to be solved to optimality. In our case this was achieved using the complete branch-and-bound method of Martello and Toth (1990a).[5] However, such runs were seen to take up to 48hrs to complete on our machines and so results were only collected for one problem instance in each class (i.e. 30 results in total). We also found that the only instances where this bound was more accurate than TMin was for the original "r" instances, whose solutions indeed seem to feature large amounts of wastage in some groups.[6]

Table 1, contrasts TMin to the smallest number of groups found in our tests (described in Section 3.5). In particular, we note that our methods have achieved TMin more regularly when tackling instances featuring larger numbers of items per group, despite the fact that such problems involve the solving of larger truss sub-problems. This suggests a further similarity to the BPP where problems are usually observed to become easier when the number of items per group is increased beyond ≈3 (Falkenauer, 1996; Martello and Toth, 1990b). This effect is lessened, however, for larger n's, perhaps demonstrating the reduced accuracy of this algorithm with larger problem instances.

3.5 Experimental Results

We tested both versions of our algorithm on the complete set of 3900 problem instances, using a time limit of 600s.[7] We experimented with various methods for producing initial solutions, including those that attempted to build up feasible groups one-by-one by pairing the closest fitting projections. Ultimately though, we found that best results in this regard were nearly always gained by simply arranging the items in descending order of area and then feeding this permutation into the first-fit algorithm. From this point onwards the algorithms then proceeded following the description given in Section 2. In particular, in each application of Local-Search, $\mathcal{T}$ was constructed by transferring each group $S_i \in \mathcal{S}$ with probability $(1/|\mathcal{S}|)$. Also, before each application of First-Fit, permutations were formed according to one of three heuristics: (a) *Fullest First*, where groups were arranged in order of decreasing spare capacity; (b) *Reverse Ordering*, where groups were arranged in the reverse order of their labelling in the previous solution; and (c) *Random Ordering*. These heuristics were applied randomly with a ratio of 5:5:3 (respectively), which has been observed to work well in research with related problems (Culberson and Luo, 1996; Lewis, 2009). We generally found the algorithms to be quite insensitive to changes in these parameters, though our settings are not claimed to be optimal.

A summary of the performance of Algorithms A and B is presented in Table 2 and, for illustrative purposes, an example solution produced by our methods is given in fig. 7. We observe in the table that the average number of groups used in the initial solutions of Algorithm B is always equal or lower than those of A, with 18 of the 30 averages being significantly different at the 5% level. We may well expect such a result since the inclusion of B&B will allow positive answers to be returned in an increased proportion of sub-problems. Indeed, this characteristic is reflected by the larger values in Algorithm B's "Succ." column, which shows that the proportion of sub-problems resulting in a positive answer increases by an average of 5.2% across the instances. We also note that the proportion of B&B applications reaching the 2s time limit averages less than 1% in all but one

[5] Fortran code available at `www.or.deis.unibo.it/knapsack.html`

[6] Note that upper bounds can also be generated using the same principles as the simplified problem: that is, simply make each item j in the TCP a rectangle of height H and width b_j, and then solve this BPP to optimality. The number of groups returned by both versions of our algorithm were seen to be less than this upper bound in all instances where the bound was generated.

[7] The algorithms were implemented in C++, and Algorithm B also used Fico Xpress optimiser version 20.00.05. All experiments were conducted on 2.83GHz Windows PCs with 3.2GB RAM.

Table 2: Results Summary. Figures appearing in bold indicate lower sample means that passed a two-tailed paired t-test at the 0.05 level

		Algorithm A (without B&B)				Algorithm B (with B&B)				
Type, n	Test Succ.[a]	Init.[b]	End[b]	Its.[c]	Succ.[d]	Init.[b]	End[b]	Its.[c]	Succ.[e]	B&B Timeout[f]
orig., a, 100	0.537	46.450	46.350	1,784,699	0.6097	46.400	46.350	965,932	0.6370	0.0000
orig., a, 200	0.535	94.950	94.750	810,708	0.5853	94.950	94.750	213,723	0.6115	0.0000
orig., a, 300	0.605	139.300	138.950	487,735	0.4687	139.250	139.100	13,523	0.5272	0.0000
orig., a, 400	0.627	183.150	182.600	323,038	0.4384	183.100	182.750	7,374	0.4842	0.0000
orig., a, 500	0.619	228.350	228.000	227,752	0.4135	228.350	228.100	3,946	0.4524	0.0000
orig., r, 100	0.469	42.529	42.163	1,722,828	0.5287	**42.508**	42.154	1,667,797	0.5650	0.0002
orig., r, 200	0.543	82.729	81.904	706,324	0.4071	**82.696**	**81.875**	691,108	0.4435	0.0001
orig., r, 300	0.540	126.483	125.279	425,008	0.3562	**126.408**	**125.242**	413,436	0.3941	0.0000
orig., r, 400	0.548	169.808	168.221	275,726	0.3315	**169.692**	**168.175**	272,406	0.3701	0.0000
orig., r, 500	0.568	212.804	210.825	197,003	0.3036	**212.600**	**210.754**	199,025	0.3392	0.0000
half, a, 100	0.480	24.100	23.750	258,913	0.4478	24.100	23.850	342	0.5214	0.0000
half, a, 200	0.628	48.600	**47.850**	90,732	0.2658	**48.300**	48.050	76	0.3133	0.0000
half, a, 300	0.687	71.700	**70.900**	48,323	0.1679	71.600	71.100	20	0.2131	0.0001
half, a, 400	0.732	94.650	**93.600**	29,205	0.1022	**94.450**	93.900	5	0.1088	0.0001
half, a, 500	0.746	118.250	**117.250**	23,557	0.0866	118.100	117.800	6	0.1180	0.0000
half, r, 100	0.459	21.300	20.796	406,413	0.3762	**21.233**	20.779	252,648	0.4531	0.0046
half, r, 200	0.595	41.425	40.475	157,806	0.1839	**41.308**	40.471	93,405	0.2376	0.0017
half, r, 300	0.642	63.167	61.729	88,112	0.1140	**63.000**	61.717	49,306	0.1502	0.0006
half, r, 400	0.652	84.263	82.463	65,900	0.0825	**84.104**	82.421	32,420	0.1162	0.0006
half, r, 500	0.666	105.663	103.250	52,818	0.0739	**105.413**	103.221	31,282	0.1017	0.0004
quart., a, 100	0.321	13.300	13.100	31,037	0.4519	13.200	13.100	10	0.5897	0.0000
quart., a, 200	0.511	26.400	**26.050**	12,099	0.2498	26.300	26.250	2	0.3472	0.0002
quart., a, 300	0.596	39.150	**38.600**	6,766	0.1357	39.100	38.800	1	0.2325	0.0009
quart., a, 400	0.599	51.400	**50.750**	3,128	0.1235	**51.200**	51.150	1	0.2298	0.0004
quart., a, 500	0.707	64.300	**63.600**	2,275	0.0377	64.200	64.200	1	0.1361	0.0011
quart., r, 100	0.280	11.779	11.554	51,877	0.4278	**11.742**	11.563	60	0.5075	0.0183
quart., r, 200	0.471	22.788	22.300	14,631	0.2021	**22.729**	22.317	1,898	0.2545	0.0075
quart., r, 300	0.542	34.533	**33.879**	9,602	0.1253	**34.488**	33.942	12	0.1608	0.0045
quart., r, 400	0.579	45.950	**45.075**	7,745	0.0913	**45.875**	45.200	181	0.1195	0.0031
quart., r, 500	0.638	57.579	**56.517**	5,855	0.0434	**57.488**	56.629	205	0.0625	0.0014

[a] Proportion of truss sub-problems solved using the tests given in Section 3.2.
[b] Number of groups in solution (mean of 1 run on each instance).
[c] Mean of one run on each instance.
[d] Proportion of greedy applications confirming $S_i \in \mathcal{F}$, averaged over all instances.
[e] Proportion of greedy+B&B applications confirming $S_i \in \mathcal{F}$, averaged over all instances.
[f] Proportion of B&B applications where the time limit of 2s was reached.

case, though there is indeed a slight increase in the quarter instances where larger sub-problems (and thus larger TSPs) are being encountered.

One interesting feature of these results is the changes that occur in the "Test" and "Succ." columns for differing values of n. For larger n's the choice of which items to assign to the same group increases which, compared to smaller instances, seems to lead to less inter-item wastage occurring in each bin. As a result, with larger instances it is less likely that further items can be added to such groups, implying that the preliminary tests (Section 3.2) are more likely to make a definite conclusion that $S_i \notin \mathcal{F}$, and also implying that the chances of "success" (via the greedy or greedy+B&B processes) will reduce. This effect is particularly prevalent with the quarter instances, where the presence of many small items often causes very little wastage to be present in many bins.

Turning our attention to results achieved at the end of a run, different patterns start to emerge with, surprisingly, Algorithm A often producing better results than B. This feature seems due to two interacting factors: the number of items per group, and whether or not the instances are of type "a" (where all items are different), or "r" (where multiple items of the same type occur). Considering the number of items per group first, we see that as this value is increased Algorithm A's use of just the greedy routines will more often result in $S_i \notin \mathcal{F}$ being concluded (when the

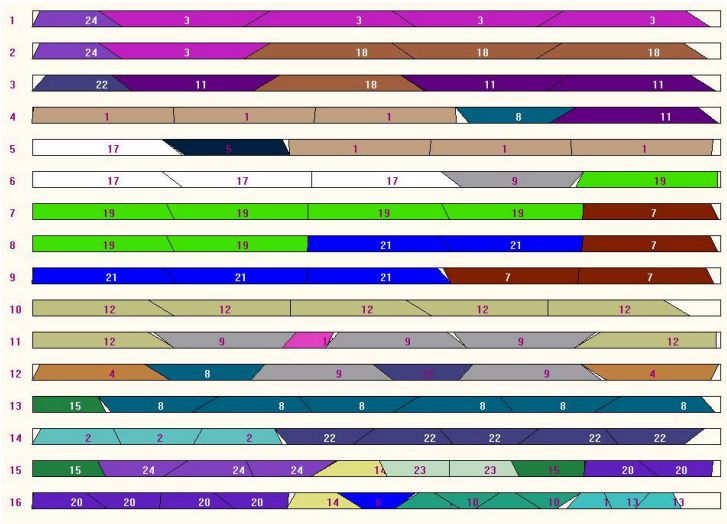


Figure 7: Example solution for a quarter "r" instance with $n = 100$. Items of the same type are labelled with the same number in the figure.

opposite is true), leading to sub-optimal configurations of items in bins. On the other hand, while Algorithm B is more accurate in this regard, the underlying sub-problems being tackled by B&B are larger, resulting in longer solution times. The upshot of this is that with more items per group, Algorithm B completes fewer iterations of the overlying local search algorithm within the time limit, resulting in a more limited search of the solution space. In conjunction with this, such effects on Algorithm B are also exacerbated when tackling larger problems because combinations of the same item types are less likely to be encountered. This reduces the advantages of using sets $\mathcal{X}$ and $\mathcal{Y}$ and is particularly apparent with the "a" instances, where sub-problems are rarely solved by referencing these sets. In contrast, when the number of items per group is small and item types are repeated, then combinations of the same item types arise more frequently, allowing many sub-problems to be solved in logarithmic time using $\mathcal{X}$ and $\mathcal{Y}$. Indeed, for the original "r" instances we see that the average number of iterations being completed by Algorithm B is always within 4% of A, ultimately bringing better results.

To illustrate the effects of these factors on the two algorithms, fig. 8 shows best-so-far profiles for four sets of problem instances with $n = 500$. For the "a" instances (top-left, bottom-left) we see that Algorithm B's use of B&B brings both slower progress and worse results at the time limit, particularly for the quarter instances (bottom-left). On the other hand, for the original "r" instances (top-right), although Algorithm B's initial progress is slower, after ≈70s it shows superior performance because sufficient information has been built up in $\mathcal{X}$ and $\mathcal{Y}$ to allow iterations to be completed at a similar rate to Algorithm A. For the quarter "r" instances, we do not witness this feature, though the figure suggests that this would eventually occur if the time limit was to be sufficiently extended.

4 Problem B: The Box Cutting Problem (BCP)

The second problem considered in this paper concerns the issue of cutting and scoring rectangles of cardboard in the construction of corrugated boxes. This problem was originally defined by Goulimis

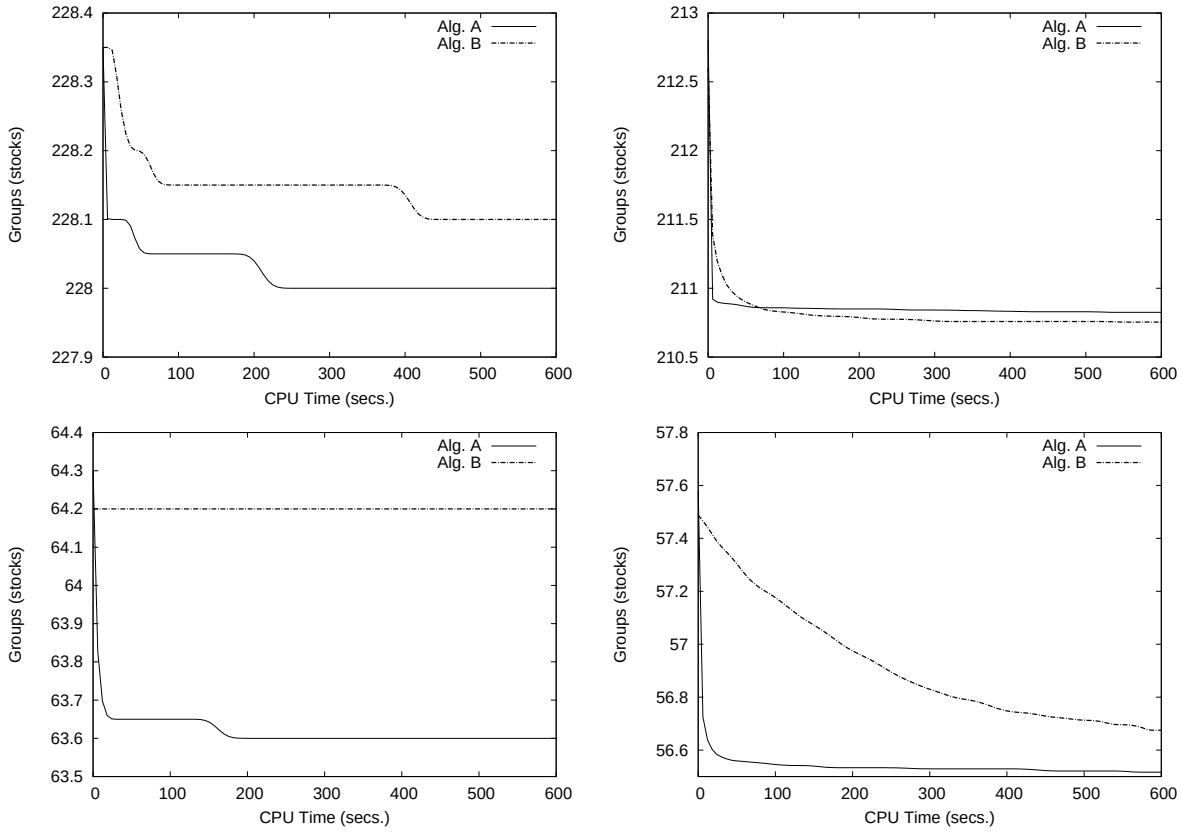


Figure 8: Best-so-far graphs using $n = 500$ for (respectively) original "a" instances, original "r" instances, quarter "a" instances, and quarter "r" instances. Each line is the average across all instances at each second.

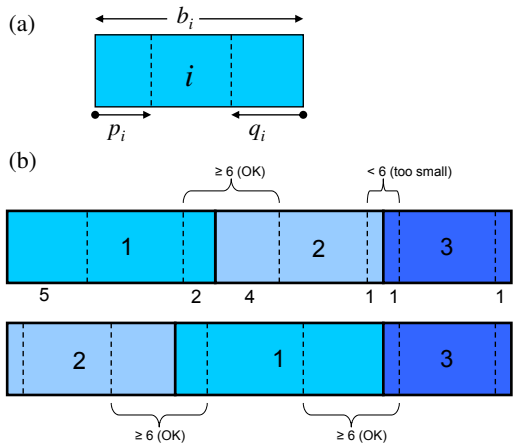


Figure 9: (a) Item dimensions for the BCP; and (b) Example of an infeasible and feasible arrangement of three items using $\delta = 6$.

(2004) who acquired the problem through an industrial contact. However, at the time of writing we are unaware of any methods for this problem that have subsequently appeared in the literature.

As with the TCP, the box cutting problem (BCP) involves two-dimensional bins (long cardboard strips) of dimensions $H \times W$, with items needing to be packed into these such that the number of bins (strips) is minimised. In this case each item $j \in \mathcal{I}$ is rectangular in shape with height H and width b_j (with $\max(b_j | j \in \mathcal{I}) \leq W$). However, it is also required that items are scored from top to bottom in two pre-determined places, allowing the cardboard to be folded along these lines at a later stage. The locations of the scores on each item are defined by two values p_j and q_j which mark the required distances of the scores from either end of the rectangle (see fig. 9(a)). For parity with the previous section, we refer to these distances as "projections".

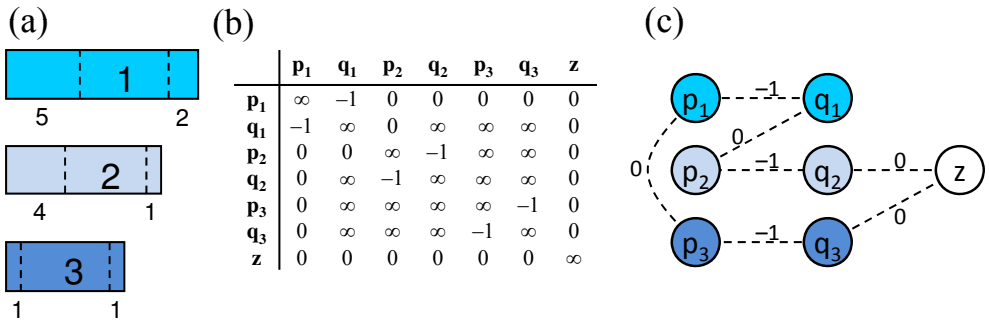
In the industrial process described by Goulimis, when items are cut from the cardboard strips, a pair of scores are simultaneously made to the left and right on the items being separated. These scores are made by a device featuring two knives that cannot be placed too close together due to their being mounted on a metal bar. This means that the distance between two successive scoring points (i.e. the total size of two adjacent projections) needs to be above a minimum scoring distance δ, which is stated by Goulimis to be about 70mm in the industrial application. To illustrate this concept, fig. 9(b) shows two arrangements of 3 items using $\delta = 6$. We observe that the first arrangement of items is infeasible since the scoring gap between items 2 and 3 is less than 6; however, the second arrangement is feasible since all scoring gaps are ≥ 6. Also note that the two outermost projections are "free" as they are not subject to the scoring distance constraint (the scores are made at a different time).

4.1 Examination of the Box Cutting Sub-problem

As with the TCP, it is clear that the ordering and orientation (on the vertical axis) of items in a bin will influence whether the arrangement is feasible. This gives rise to the following sub-problem:

Definition 4.

The box sub-problem asks: given a bin of size $H \times W$, and $S_i \subseteq \mathcal{I}$ such that $A(S_i) \leq HW$, is $S_i \in \mathcal{F}$?



	p_1	q_1	p_2	q_2	p_3	q_3	z
p_1	∞	−1	0	0	0	0	0
q_1	−1	∞	0	∞	∞	∞	0
p_2	0	0	∞	−1	∞	∞	0
q_2	0	∞	−1	∞	∞	∞	0
p_3	0	∞	∞	∞	∞	−1	0
q_3	0	∞	∞	∞	−1	∞	0
z	0	0	0	0	0	0	∞

Figure 10: Example conversion of a box sub-problem to a TSP. In (c) the route shown has a total distance of $-|S_i| = -3$ and corresponds to the feasible configuration of the items in fig. 9(b)).

In similarity to the TCP, we observe that this sub-problem can also be reduced to a corresponding TSP. This is achieved by again considering each projection as a city, with distances between these being calculated as follows:

- distances between projections on the same item are set to -1;
- distances between projections on different items whose total size is less than δ are set to ∞;
- all other inter-city distances are set to zero.

It is also necessary to use one dummy city in this model which is to be connected to all other cities using arcs of distance zero. Fig. 10 shows an example of this conversion using $\delta = 6$ and $|S_i| = 3$. It is clear that $S_i \in \mathcal{F}$ if and only if a Hamiltonian cycle of distance $-|S_i|$ exists (since only routes of this distance represent a feasible configuration of the items). We also observe that a larger value of δ here would see more arcs assuming distances of ∞, making the problem less likely to be solvable. For example, increasing δ from 6 to 7 in fig. 10 results in distances of ∞ also being assumed between p_1 and p_3, and p_1 and q_3, meaning that no Hamiltonian cycle of distance $-|S_i|$ exists.

4.2 Addressing the Box Cutting Sub-Problem

Our method for the box sub-problem follows a similar strategy to the TCP algorithm: preliminary tests followed by a greedy algorithm, and then finally a more sophisticated branch-and-bound procedure (if desired). As before, let $S_i \subseteq \mathcal{I}$ be a group of items such that $A(S_i) \leq HW$. In addition, let $\mathbf{v} = (v_1, v_2, \ldots, v_{2|S_i|})$ be a vector of the items' projections in ascending order of size. The preliminary tests we conduct are as follows:

- If $|S_i| = 1$ then $S_i \in \mathcal{F}$.
- If $v_3 \geq \delta/2$, and v_1 and v_2 occur on different items, then $S_i \in \mathcal{F}$. In this case a feasible arrangement of the items is guaranteed since we can place the two smallest projections, v_1 and v_2, in the outermost positions, and all other scoring gaps will be greater or equal to δ.
- If $v_3 + v_{2|S_i|} < \delta$, then $S_i \notin \mathcal{F}$. Assuming v_1 and v_2 are to be placed in the outermost positions, if the next largest projection v_3 is too small to be paired with the largest projection $v_{2|S_i|}$, then obviously we cannot hope to find a feasible arrangement of the items.

As before, if these checks prove inadequate, a greedy heuristic is next applied that attempts to place all items in S_i into the bin. At each iteration this is done by considering the currently unplaced items and choosing the one with the smallest feasible projection,[8] breaking ties by selecting the item that has the largest opposite projection. (The bottom arrangement in fig 9(b) is a result of this process.) This heuristic has the effect of pairing projections whose combined sizes are "just large enough", encouraging the largest projections to be reserved for pairing with the smallest. It also ensures that the smallest projection is placed on the leftmost position in the bin, eliminating it from the sub-problem.

Similarly to our method for the TCP, if this greedy heuristic fails to find a feasible configuration of the items, we also have the option of invoking more powerful methods for determining membership of $\mathcal{F}$. In our case this is again achieved with Xpress using the same formulation as Section 3 (Appendix A), but also with the added constraint that the returned solution must feature an objective function (eq. (8)) of *exactly* $-|S_i|$. As before, we also maintain sets $\mathcal{X}$ and $\mathcal{Y}$ during runtime to ensure that the B&B procedure is not used more than once on a particular combination of items types.

Finally, as mentioned in Section 1, one important observation of the box sub-problem is that condition (4) is not satisfied. That is, if we have a set of items $S_i \in \mathcal{F}$, then the removal of one or more of these items will not necessarily preserve feasibility (observe, for example, the effects of eliminating item 1 from fig. 9). This feature is in contrast to the TCP and implies that an application of First-Fit using a permutation formed in the manner described in Section 2 could result in a solution that features more groups than a previous solution. In our case we do not consider this to be a major problem as First-Fit still seems to be able to carry out its primary purpose of escaping local optima. In experiments we also observed that the additional groups that could result from this procedure were usually eliminated quite quickly by subsequent applications of Local-Search. However, this is an interesting feature of the BCP and it would be interesting for future research to investigate other possibilities here.

4.3 Experimental Results

Information regarding typical industrial item sizes and score distances is not readily available for the BCP, and we are compelled to produce artificial problem instances at this point. For our experiments, we took the well-known one-dimensional SCP instance "10a" from the work of Liang et al. (2002), which involves $n = 600$ items of 36 types with $\sum_{j \in \mathcal{I}} A(j) = 25,790$. This is seen to be a difficult problem and, by default, uses $W = 120$ and $H = 1$ resulting in approx. 2.8 items per group in the known optimal of 215 groups. We then modified this instance by adding projections to each of the 36 item types. This was achieved by randomly selecting p_j and q_j from the interval $[1, \frac{b_j}{2}]$, ensuring that the distance between these $b_j - (p_j + q_j)$ (i.e. the "central width" of the item) was always greater than 10% of b_j.

For our experiments two problem features in particular were considered: the number of items per group, and the parameter α, which gives the proportion of projection pairs (from different items) whose total size is $> \delta$. The former can be controlled by simply altering W, while the latter was controlled by making alterations to δ such that values for α from 0.0 to 1.0 (with increments of 0.1) were determined. Obviously, if $\alpha = 0.0$ then the number of groups required will be n, while $\alpha = 1.0$ implies that all projection pairs are $> \delta$, making the BCP equivalent to the original SCP instance.

[8] That is, the smallest projection that, when paired with the rightmost projection of the previously inserted piece, results in a scoring distance $\geq \delta$. When selecting the first item to place into the bin, all projections are feasible and so the smallest projection overall is selected

Table 3: Results Summary. Figures in bold indicate lower sample means that passed a two-tailed paired t-test at the 0.05 level. TMin for $W = 120$, 240, and 360 = 215, 108, and 72 respectively.

			Algorithm A (without B&B)			Algorithm B (with B&B)			
α, W	Items per group	Test Succ.[a]	End[b]	Its.[c]	Succ.[d]	End[b]	Its.[c]	Succ.[e]	$\mathcal{X}$, $\mathcal{Y}$ Used[f]
0.0, 120	1.000	1.000	600.00	4,475	-	600.00	4,475	-	-
0.1, 120	1.367	0.967	439.00	9,846	0.071	439.00	10,457	0.071	1.00000
0.2, 120	1.852	0.805	323.95	13,157	0.059	323.95	16,864	0.059	1.00000
0.3, 120	2.360	0.560	255.75	18,441	0.075	**254.25**	27,679	0.078	1.00000
0.4, 120	2.727	0.380	220.60	73,654	0.317	**220.00**	92,118	0.329	1.00000
0.5, 120	2.778	0.296	216.00	147,387	0.665	216.00	147,128	0.686	1.00000
0.6, 120	2.778	0.344	216.00	203,532	0.882	216.00	185,618	0.899	0.99999
0.7, 120	2.778	0.478	216.00	224,732	0.931	216.00	205,138	0.931	0.99999
0.8, 120	2.778	0.677	216.00	250,850	0.978	216.00	236,167	0.978	0.99998
0.9, 120	2.778	0.807	216.00	272,035	0.997	216.00	261,498	0.997	0.99992
1.0, 120	2.778	1.000	216.00	299,706	-	216.00	299,817	-	-
0.0, 240	1.000	1.000	600.00	4,265	-	600.00	4,260	-	-
0.1, 240	1.367	0.975	439.00	7,219	0.135	439.00	7,550	0.171	0.99997
0.2, 240	1.892	0.929	**317.05**	9,021	0.256	317.35	6,844	0.269	0.99997
0.3, 240	2.552	0.860	**235.10**	11,134	0.355	235.80	3,383	0.361	0.99988
0.4, 240	3.891	0.557	**154.20**	9,848	0.397	158.20	1,041	0.424	0.99849
0.5, 240	5.533	0.215	108.60	8,860	0.492	108.85	816	0.503	0.99733
0.6, 240	5.556	0.174	108.00	40,864	0.626	108.00	1,846	0.643	0.99469
0.7, 240	5.556	0.203	108.00	86,443	0.811	108.00	4,151	0.822	0.99112
0.8, 240	5.556	0.343	108.00	127,040	0.942	108.00	23,203	0.942	0.99710
0.9, 240	5.556	0.547	108.00	157,867	0.994	108.00	77,817	0.995	0.97842
1.0, 240	5.556	1.000	108.00	235,675	-	108.00	235,322	-	-
0.0, 360	1.000	1.000	600.00	4,265	-	600.00	4,261	-	-
0.1, 360	1.367	0.973	439.00	6,704	0.118	439.00	6,360	0.134	0.99979
0.2, 360	1.893	0.941	**317.00**	8,074	0.332	317.55	1,094	0.343	0.99878
0.3, 360	2.564	0.838	**234.05**	7,537	0.422	235.65	342	0.427	0.99757
0.4, 360	4.028	0.574	**148.95**	5,355	0.427	154.50	127	0.446	0.98789
0.5, 360	7.381	0.235	**81.30**	2,630	0.486	86.90	80	0.498	0.99127
0.6, 360	8.333	0.106	72.00	6,512	0.528	72.00	158	0.531	0.99642
0.7, 360	8.333	0.124	72.00	35,755	0.723	72.00	695	0.730	0.99208
0.8, 360	8.333	0.183	72.00	73,470	0.932	72.00	4,612	0.932	0.99745
0.9, 360	8.333	0.347	72.00	92,825	0.995	72.00	25,892	0.995	0.93601
1.0, 360	8.333	1.000	72.00	172,566	-	72.00	172,748	-	-

[a]Proportion of box sub-problems solved using the tests given in Section 4.2.
[b]Number of groups in final solution (mean of 20 runs).
[c]Mean of 20 runs.
[d]Proportion of greedy applications confirming $S_i \in \mathcal{F}$, averaged over all runs.
[e]Proportion of greedy+B&B applications confirming $S_i \in \mathcal{F}$, averaged over all runs.
[f]Proportion of times the box sub-problem was solved using $\mathcal{X}$ or $\mathcal{Y}$ as opposed to running B&B

Table 3 shows the results gained from 20 runs of the two algorithms for various values of α and W using the same experimental conditions as Section 3. For very low and high values of α, the associated sub-problems are usually solved using the initial tests, resulting in similar or identical performance of the two algorithms. As expected, we also observe that for low α-values many groups are needed, while for large α-values (where nearly all combinations are feasible) the problem is similar to the original problem instance. Indeed, for $W = 240$ and 360, TMin has been achieved in all cases for $\alpha \geq 0.6$.

For more central values of α, where the tests are not usually sufficient for solving the sub-problems, Algorithm B again offers a slight increase in the proportion of sub-problems resulting in a positive answer (up to 3.4%), indicating a higher accuracy in this regard. However, the advantages that B&B brings are again only apparent in cases where the number of items per group is small (here, when $W = 120$). For the remaining cases, the larger sub-problems result in Algorithm B completing far fewer iterations within the time limit, allowing Algorithm A to return significantly better results. Note that these effects would also be accentuated were the number of item types to

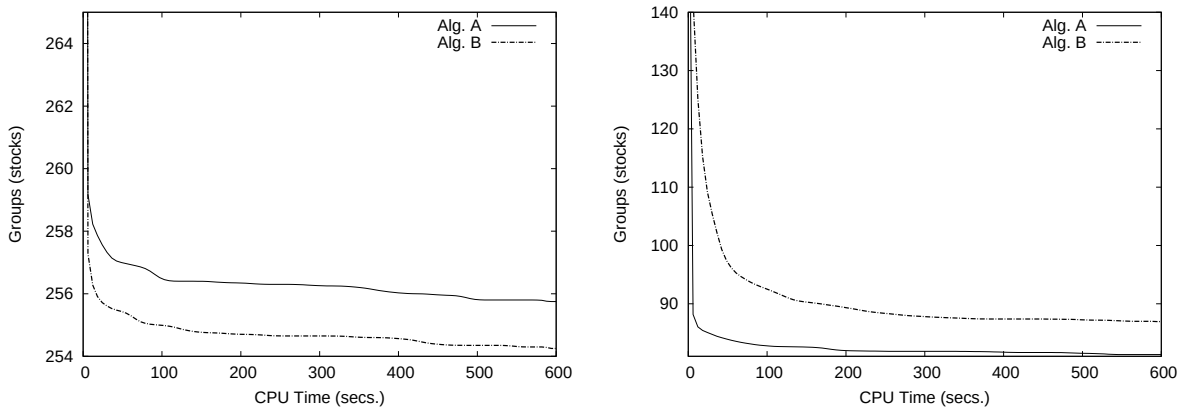


Figure 11: Best-so-far graphs for (respectively) $\alpha = 0.3$, $W = 120$ and $\alpha = 0.5$, $W = 360$. Each line is the average across all runs at each second.

be increased to approach n as the advantages of using $\mathcal{X}$ and $\mathcal{Y}$ would also lessen.

To further illustrate these features, fig. 11 shows best-so-far profiles for two combinations of W and α. In the first case the sub-problems are small (average of 2.36 items per group), and Algorithm B shows superior performance over time. On the other hand, the second example involves much larger sub-problems (average of 7.38 items per group) hence B's progress is significantly slowed, allowing Algorithm A to produce the best results within the time limit.

5 Conclusions and Further Work

This paper has examined two new bin packing-based problems for which two fundamental questions need to be considered: (a) which items should be assigned to each bin? and (b) how should the items be arranged in each bin? It is apparent that when using an optimisation algorithm such as ours, many instances of problem (b) will be faced during a run. However, our results indicate that if too much effort is spent on trying to find optimal (or near optimal) arrangements, then this can lead to inadequate amounts of effort being applied to problem (a), resulting in solutions of poorer quality. Indeed, even under a rather generous time limit of 600s, we have generally observed that simple greedy heuristics for approximating (b) have resulted in solutions superior to those achieved via a more accurate but costly branch-and-bound method. On the other hand, we have also seen that our use of sets $\mathcal{X}$ and $\mathcal{Y}$ for collecting information on previous applications of branch-and-bound can be useful, particularly when many items in $\mathcal{I}$ are of the same type. Indeed, if in a practical setting it is the case that only a limited number of different item types are ever considered, it would be beneficial to simply save permanently the contents of $\mathcal{X}$ and $\mathcal{Y}$, making future runs of the algorithm far more efficient.

Perhaps the most obvious avenue of future research with these problems will be in the design of improved methods for approximating/solving the underlying sub-problems, particularly if they prove to be more accurate than our current greedy heuristics, but also less expensive than our branch-and-bound technique. For instance, we could investigate whether commercial solvers (such as Xpress and CPLEX) perform better with other integer programming formulations of the underlying TSP (see, for example, (Padberg and Sung, 1991)), or we could simply create new techniques that exploit the very specific structures of the sub-problems. It would also be interesting to see

how other existing methods for the SCP and BPP might be adapted to these problems, including metaheuristic approaches (Falkenauer, 1998; Leung et al., 1997; Levine and Ducatelle, 2003) and others (Haessler and Sweeny, 1991; Roodman, 1986).

It would also be desirable for future research to examine methods that are more flexible with regards to variants of this problem that might also arise in industry. For example, with the truss cutting problem, in addition to minimising the number of bins needed to pack the n items, it might be preferable for a solution's wastage to occur in large contiguous blocks wherever possible, because these "left-over" pieces of material might then be used in a future problem. One possibility here would be to evaluate candidate solutions using a cost measure such as the one suggested by Falkenauer (1998) for the BPP:

$$f = \frac{\sum_{S_i \in \mathcal{S}} (\frac{A(S_i)}{HW})^2}{|\mathcal{S}|} \tag{6}$$

which, when being maximised, differentiates between solutions using the same number of bins by favouring those that feature "extreme" (i.e. near-full and near-empty) bins. In addition, if different sized bins are to be used in a particular problem, this means the algorithm will also be concerned with deciding which sized bins should be used for the containing each of the items. Thus factors relating to the variable sized bin packing problem will also be relevant here (Alves and Valerio de Carvalho, 2007; Haouari and Serairi, 2009; Kang and Park, 2003).

One further aspect of the TCP is the question of how it might be classified according to accepted taxonomies of packing and cutting problems. We noted in Section 1 that, according to Wäscher et al. (2007), the TCP falls into the class of fixed-dimension input-minimisation problems. However, what is less clear is whether the problem is best classified as a 1D or 2D problem. Note that the height of items and bins in the TCP is fixed, meaning it can essentially be discounted by the solution process. This might suggest that the problem is 1D, although it is clearly different to the classical 1D BPP due to the item ordering and orientation considerations. Perhaps then it is better to consider the TCP as a special case of 2D problem since this classification allows us to capture the problem feature of inter-item wastage caused by the items' projections.

Concluding, from our perspective it is slightly unfortunate that we were unable to get hold of any real-world data sets for these problems, though our artificially generated instances have at least allowed us to gauge the circumstances under which our proposed methods seem to succeed and fail. To aid further investigation in this area we have posted the complete set of 3901 problem instances online at `www.rhydlewis.eu/resources/trapBoxProbs.zip`. We invite other researchers to download these for use in their own investigations.

References

Aardel, K. I., van Hoesel, S. P. M., Koster, A. M. C. A., Mannino, C., and Sassano, A. (2002). Models and solution techniques for the frequency assignment problems. *4OR : Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(4):1–40.

Alves, C. and Valerio de Carvalho, J. (2007). Accelerating column generation for variable sized bin-packing problems. *European Journal of Operational Research*, 183:1333–1352.

Culberson, J. and Luo, F. (1996). Exploring the k-colorable landscape with iterated greedy. In Johnson, D. S. and Trick, M. A., editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26, pages 245–284. American Mathematical Society.

de Werra, D. (1988). Some models of graphs for scheduling sports competitions. *Discrete Applied Mathematics*, 21:47–65.

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 and Sons.

Garey, M. and Johnson, D. (1979). *Computers and Intractability - A guide to NP-completeness*. W. H. Freeman and Company, San Francisco, first edition.

Goulimis, C. (2004). Viewpoint: Minimum score separation — an open combinatorial problem associated with the cutting stock problem. *Journal of the Operational Research Society*, 55:1367–1368.

Haessler, R. and Sweeny, P. (1991). Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54:141–150.

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

Hertz, A., Plumettaz, M., and Zufferey, N. (2008). Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560.

Isomoto, K., Wakabayashi, S., Yoshida, N., and Miyao, J. (1993). A parallel algorithm for k-way graph partitioning. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 76(3):23–33.

Jensen, T. R. and Toft, B. (1994). *Graph Coloring Problems*. Wiley-Interscience, 1 edition.

Kang, J. and Park, S. (2003). Algorithms for the variable sized bin packing problem. *European Journal of Operational Research*, 147:365–372.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. and Thatcher, J., editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press.

Kendall, G., Knust, S., Ribeiro, C., and Urrutia, S. (2010). Scheduling in sports, an annotated bibliography. *Computers and Operations Research*, 37(1):1–19.

Leung, Y., Gao, Y., and Xu, Z. (1997). Degree of population diversity - a perspective on premature convergence in genetic algorithms and its marcov chain analysis. *IEEE Trans. Neural Networks*, 8(5):1165–1765.

Levine, J. and Ducatelle, F. (2003). Ant colony optimisation and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55(12)(7):705–716.

Lewis, R. (2008). A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30(1):167–190.

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 and Operations Research*, 36(7):2295–2310.

Liang, K., Yao, X., Newton, C., and Hoffman, D. (2002). A new evolutionary approach to cutting stock problems with and without contiguity. *Computers and Operations Research*, 29:1641–1659.

Martello, S. and Toth, P. (1990a). *Knapsack Problems: Algorithms and Computer Implementations*. Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Chichester-New York.

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

McCollum, B., Schaerf, A., Paechter, B., McMullan, P., Lewis, R., Parkes, A., L., D. G., Qu, R., and E.‘, B. (2010). Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130.

Miller, C., Tucker, A., and Zemlin, R. (1960). Integer programming formulations and traveling salesman problems. *Journal of the ACM*, 7:326–329.

Nakano1, S., Zhou1, X., and Nishizeki1, T. (1995). *Computer Science Today*, chapter Edge-coloring algorithms, pages 172–183. Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

Padberg, M. and Sung, T. (1991). An analytical comparison of different formulations of the travelling salesman problem. *Mathematical Programming*, 52:315–357.

Rasmussen, R. and Trick, A. (2008). Round robin scheduling – a survey. *European Journal of Operational Research*, 188:617–636.

Roodman, G. (1986). Near-optimal solutions to one-dimensional cutting stock problems. *Computers and Operations Research*, 13:713–719.

Valenzuela, C. M. (2001). A study of permutation operators for minimum span frequency assignment using an order based representation. *Journal of Heuristics*, 7:5–21.

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

A Integer Programming Formulation of the Truss and Box Cutting Sub-Problems

The integer programming formulation of the underlying TSP for both the truss and box sub-problems was based on the classical Miller-Tucker-Zemlin model (Miller et al., 1960). In either case let D be a distance matrix of m cities, constructed in the manner described in Section 3.1 or 4.1. Now let x_{ij}, $(i = 1, \ldots, m,\ j = 1, \ldots, m)$ be a binary variable such that:

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i,j) \text{ is included in the tour} \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The cost of a tour (total distance travelled) is calculated as:

$$\sum_{i=1}^{m} \sum_{j=1}^{m} x_{ij} D_{ij} \tag{8}$$

subject to:

$$\begin{array}{ll} \sum_{j=1}^{m} x_{ij} = 1 & \forall i \\ \sum_{i=1}^{m} x_{ij} = 1 & \forall j \\ x_{ii} = 0 & \forall i \end{array} \tag{9}$$

Eq. (9) ensures that each of the m cities is visited exactly once in a tour. In addition, it is also necessary to exclude the possibility of sub-tours (that is, the defined route must be a Hamiltonian cycle); thus an additional integer variable u_i $(i = 1, \ldots, m)$ is introduced with constraints:

$$\begin{array}{ll} u_1 = 1, & \\ 2 \leq u_i \leq m & \forall i \neq 1, \\ u_i - u_j + 1 \leq (m-1)(1 - x_{ij}) & \forall i \neq 1,\ \forall j \neq 1. \end{array} \tag{10}$$

Finally, we also define the constraint:

$$x_{2k,2k-1} + x_{2k-1,2k} = 1 \quad \forall k, \tag{11}$$

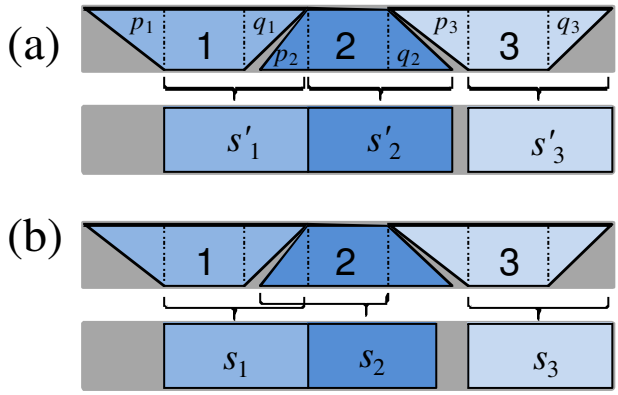


Figure 12: Producing a simplified problem from the a corresponding TCP

(where $k = 1, \ldots, (m/2)$ and $k = 1, \ldots, ((m-1)/2)$ for the truss and box sub-problems respectively). This specifies the number of $-\infty$ arcs (respectively (-1) arcs) that are required for a tour to correspond to a legal configuration of the items.

B Proof of Theorem 1

To prove Theorem 1 it is sufficient to show that for any feasible configuration of items in a bin in the TCP, a corresponding feasible configuration of items exists in the simplified problem.

Assume the left- and right-most projections of an item j are of size p_j and q_j respectively. If we were to eliminate the left-most projection of all items j, we would obtain simplified rectangular items of width $s'_j = b_j - p_j$ and height H. Obviously, these simplified items could be feasibly packed into an $H \times W$ bin by simply using the same item-ordering as the original pattern (see fig. 12(a)).

However, in our case there are two possibilities for each item j: $p_j \geq q_j$ or $p_j < q_j$. In the first case, we have $s'_j = b_j - p_j = b_j - \max(p_j, q_j) = s_j$, otherwise we have $s'_j = b_j - p_j \geq b_j - \max(p_j, q_j) = s_j$. Thus, in any feasible solution of the TCP, the corresponding rectangular items of width $s_j = b_j - \max(p_j, q_j)$ and height H can be packed into the same groups.□

Fig. 12(b) gives a small example. Note that since $p_3 \geq q_2$, there is a gap between the simplified widths $s_2 = b_2 - \max(p_2, q_2) = b_2 - p_2$ and $s_3 = b_3 - \max(p_3, q_3) = b_3 - p_3$. This indicates that the simplified width of item 3 could in fact be larger than $b_3 - p_3$. However, this is not the case for all the items. Thus the simplified width is tight, which means no larger width can be used as the simplified width in order to gain a lower bound of the original problem.