Discrete Optimization 6 (2009) 174–188

Contents lists available at ScienceDirect

Discrete Optimization

journal homepage: www.elsevier.com/locate/disopt

A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem

N. Ntene, J.H. van Vuuren *

Department of Logistics, University of Stellenbosch, Private Bag X1, Matieland, 7602, South Africa

ARTICLE INFO

Article history:
Received 1 February 2006
Received in revised form 8 September 2008
Accepted 22 November 2008
Available online 22 January 2009

Keywords:
Strip packing problem
Level algorithms

ABSTRACT

An overview and comparison is provided of a number of heuristics from the literature for the two-dimensional strip packing problem in which rectangles have to be packed without rotation. Heuristics producing only guillotine packings are considered. A new heuristic is also introduced and a number of modifications are suggested to the existing heuristics. The resulting heuristics (known and new) are then compared statistically with respect to a large set of known benchmarks at a 5% level of significance.

1. Introduction

The objective in packing problems is to determine an optimal arrangement of items of specified dimensions such that some cost function (typically measuring spatial wastage) is minimised. It is usually not possible to find optimal solutions to large instances of such problems within a reasonable amount of time, because packing problems are generally NP-hard [6,10,22]. Various methods have been proposed to solve a variety of packing problems (exactly or approximately) and these methods may be grouped into the three broad classes: *heuristics* [12,24], *meta–heuristics* [12,14] and *exact methods* [22]. Heuristics are methods based on intuitive and/or plausible arguments that give good solutions under certain conditions, but do not guarantee optimality [25]. Meta–heuristics are approximate procedures that efficiently and effectively search through the solution space (or a subspace thereof) heuristically, by iteratively employing a collection of (possibly greedy) heuristics. Finally, exact methods are guaranteed to find optimal solutions.

Two-dimensional packing problems occur in a variety of different types—some of the most common types include *strip packing problems* [8,13,14,16–18], *bin packing problems* [8,14,18], *knapsack problems* [9] and *cutting stock problems* [11,17]. Strip packing problems (also referred to as *open dimension problems* [31]) involve packing items into a single bin (referred to as a *strip*) of fixed width W and infinite height, with the objective of minimising the total height of the packing within the strip [3]. Bin packing problems, on the other hand, involve packing items into multiple bins of fixed width and height, so as to minimise the number of bins utilised. Knapsack problems involve packing a number of items, each with an associated utility value, into a bin, with the objective of maximising the combined utility value of items packed. Cutting stock problems are dual or cutting counterparts of packing problems and involve allocating a weakly heterogeneous[1] set of items to a selection of large objects of minimal value [31]. In all the different types of packing problems, the items packed are not allowed to overlap, while the orientation of items to be packed may or may not be fixed, depending on the application.

In this paper, we focus on strip packing problems, in which the items to be packed are rectangular and have a fixed orientation. An example of an application of this type of problem is encountered in the paper industry where raw materials are typically in the form of fixed-width rolls of paper (which may be considered of infinite length for all practical purposes)

* Corresponding author.
E-mail addresses: ntene@dip.sun.ac.za (N. Ntene), vuuren@sun.ac.za (J.H. van Vuuren).

[1] Wascher [31] describes weakly heterogeneous items as a set of items that may be grouped into relatively few classes of identical size and shape.

1572-5286/$ – see front matter
doi:10.1016/j.disopt.2008.11.002

Table 1
Dimensions of rectangles used as example instance in Sections 2–4.

	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}	L_{11}	L_{12}
$h(L_i)$	6	16	20	24	4	4	6	16	4	6	3	3
$w(L_i)$	8	32	3	24	13	2	7	11	8	12	13	28

and where the aim is to find an arrangement of printed items that results in a minimum paper length utilised [4,14,19,22]. Many other applications also exist [1,17–20,29].

Some of the most basic and most commonly used heuristics for strip packing problems may be grouped into the classes of *level algorithms, shelf algorithms and plane algorithms*. The first class of algorithms is primarily used to solve packing problems commonly known as *offline* packing problems, in which the entire list of rectangles to be packed is known in advance. The algorithms typically partition the strip into horizontal levels, with the bottom of the strip representing the first level, and then proceed to pack rectangles onto these levels. Shelf algorithms, on the other hand, are generally used to solve packing problems in which the dimensions of the next rectangle to be packed only become known once the current rectangle has been packed (these problems are referred to as *online* packing problems). Shelf algorithms also divide the strip into horizontal shelves — however, the shelf heights are chosen relatively larger than necessitated by the partial packing that has been realised, so as to create sufficient space, on expectation, for taller rectangles that may occur later in the packing list. In plane algorithms, the strip is not partitioned and rectangles may be placed in any available space where they fit within the strip.

Our aim in this paper is to review and evaluate packing heuristics in the literature from the class of *level algorithms* producing guillotine packings[2] for offline problem instances, and to propose variations to some of these procedures. We start our exposition with a brief description of six known heuristics in Section 2, followed by our proposed variations in Section 3 to some of these heuristics. An entirely new heuristic is presented in Section 4, followed, in Section 5, by a description of a large set of benchmark problems that we used to test and compare algorithm efficiencies. All the heuristics (old and new) were computer-implemented in Visual Basic 6 [23], so that their performances could be compared with respect to the benchmark instances. The algorithmic results are presented in Section 6. Some final remarks follow in Section 7. The notation used throughout the paper is that a list of n rectangles $\mathscr{L} = \{L_1, L_2, \ldots, L_n\}$ is to be packed into a strip of width W. The width and height of rectangle L_i in the list $\mathscr{L}$ is denoted by $w(L_i)$ and $h(L_i)$ respectively.

2. Known level algorithms

In all *level algorithms*, the rectangles in $\mathscr{L}$ are placed with their lower edges on certain horizontal levels within the strip. The height of each level is determined by the heights of the tallest rectangles placed on previous levels. In this section we consider six standard level algorithms from the literature and propose a total of six new variations on some of these algorithms. The mechanisms of the resulting twelve algorithms are illustrated by means of a simple example instance of the strip packing problem, which requires twelve rectangles (see Table 1) to be packed into a strip of width 40 units.

2.1. The next-fit decreasing height algorithm

In the so-called *next-fit decreasing height* (NFDH) algorithm [6], the list of rectangles to be packed is pre-ordered according to non-increasing height. This implies that the first rectangle placed on a level determines the height of the next level. A rectangle is placed on the current level, left justified, if it fits. However, if it does not fit, then a new level is created above the current level (which becomes the new current level) and the rectangle is placed there, left justified. The packing progresses from left to right per level and from the bottom of the strip upwards level-wise. Levels lower than the current level are never revisited. Rectangles of equal height retain their original order in the packing list relative to each other. This procedure achieves a packing of height 56 units for our example instance in Table 1, as shown in Fig. 1(a).

2.2. The first-fit decreasing height algorithm

In the so-called *first-fit decreasing height* (FFDH) algorithm [6], the list of rectangles is also pre-ordered according to non-increasing height. Rectangles of equal height again retain their original order relative to each other in the packing list. A rectangle is placed left justified on the *lowest* level with sufficient space. The strip is searched level-wise from the bottom upwards for sufficient packing space, and if the current rectangle does not fit into any of the existing levels, a new level is created above the current top level (which becomes the new top level) and the rectangle is placed there, left justified. The difference between the NFDH and FFDH algorithms is, therefore, that in the latter, previously packed levels are always searched for sufficient space to pack a rectangle whereas in the former, previously packed levels may not be revisited. This procedure results in a packing height of 53 units for our example instance in Table 1, as illustrated in Fig. 1(d).

[2] Packings which may be disseminated by performing a series of edge-to-edge cuts.

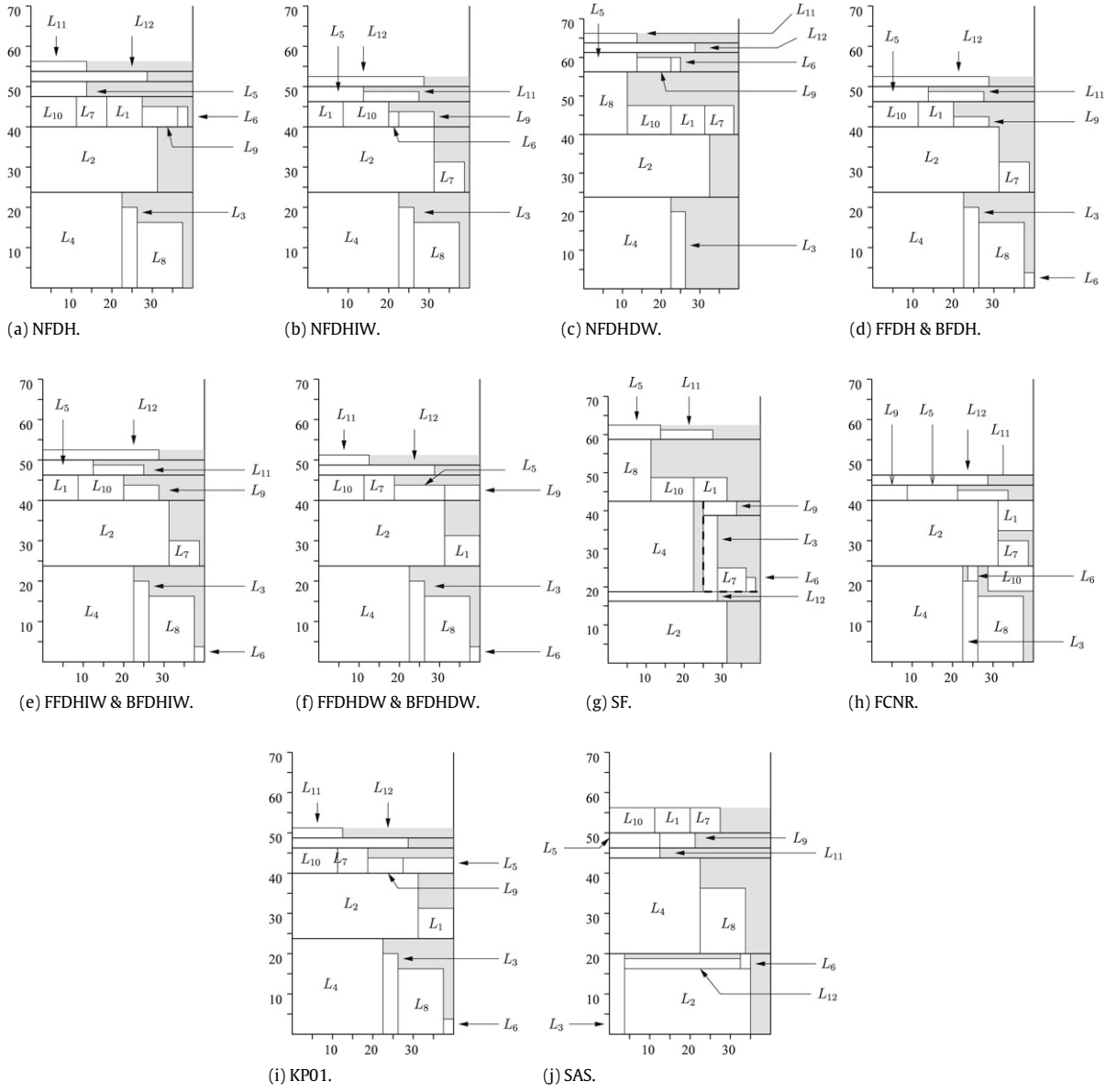

Fig. 1. Packings produced by the level algorithms described in Sections 2–4 for the example instance of the strip packing problem specified in Table 1.

2.3. The best-fit decreasing height algorithm

The *best-fit decreasing height* (BFDH) algorithm [24] is analogous to the FFDH algorithm, except that in this algorithm rectangles are placed left justified towards the right of the last rectangle packed on the level with *minimum residual horizontal space*. This means that to pack the next rectangle, all existing levels are searched for sufficient space and the area of the horizontal space towards the right of the level packing that would remain un-utilised if the rectangle were to be placed in any of the levels, is computed. The rectangle is placed on the level that leaves the smallest horizontal space. When this algorithm is applied to our example instance in Table 1, a total packing height of 53 units is again obtained, as shown in Fig. 1(d).

2.4. The split fit algorithm

In the *split fit* (SF) algorithm [6] the lengths and widths of all rectangles to be packed are scaled so that the strip has unit width. The largest integer $m \geq 1$ is then determined for which all rectangles in $\mathscr{L}$ have width less than or equal to $1/m$. The list is divided into two sub-lists $\mathscr{L}_{wide}$ and $\mathscr{L}_{narrow}$, both ordered according to non-increasing height such that $\mathscr{L}_{wide}$ contains all rectangles whose widths are greater than $1/(m+1)$ and $\mathscr{L}_{narrow}$ contains all rectangles whose widths are at most

$1/(m+1)$. The rectangles in the list $\mathscr{L}_{wide}$ are packed using the FFDH algorithm and all the rectangles placed on a particular level are referred to collectively as a *block*. The blocks of this packing are then rearranged such that blocks of total width greater than $(m+1)/(m+2)$ are at the bottom of the packing, followed by blocks of total width at most $(m+1)/(m+2)$. This process of shifting the blocks creates a rectangular region $\mathscr{R}$ of width $1/(m+2)$ to the right of the latter blocks. The rectangles in $\mathscr{L}_{narrow}$ are then packed, again using the FFDH algorithm, with the packing starting in the region $\mathscr{R}$. If a rectangle does not fit into $\mathscr{R}$, then the packing continues above the packing of $\mathscr{L}_{wide}$.

In our example instance in Table 1 (after rescaling the rectangle dimensions) a value of $m = 1$ is used. This ensures that all rectangles have width at most 1. The list of wide rectangles (rectangles of width greater than $1/2$) comprises rectangles L_2, L_4 and L_{12} which are packed using the FFDH algorithm. The remaining nine rectangles in the narrow list are then packed in the region $\mathscr{R}$ provided there is sufficient space, or else above the packing of the wide rectangles. The SF algorithm achieves a packing height of 63 units when applied to our example instance, as illustrated in Fig. 1(g).

2.5. The floor-ceiling no rotation algorithm

In the *floor-ceiling no rotation* (FCNR) algorithm [19] rectangles are pre-ordered according to non-increasing height. Within a level, a *floor* is defined as the horizontal line coinciding with the bottom edges of rectangles packed on that level, while a *ceiling* is a horizontal line coinciding with the upper edge of the tallest rectangle packed on that level. If there is sufficient space to accommodate a rectangle on a floor, then the rectangle is said to be *floor feasible*. Rectangles are packed on the floor from left to right, their left-hand edges coinciding with the right-hand edges of previously packed rectangles. The first rectangle to be placed on a ceiling is packed with its right-hand edge coinciding with the right-hand edge of the strip and the level is said to be *ceiling-initialised*. Rectangles are packed on a ceiling from right to left (however, empty spaces may intentionally be left between consecutive rectangles to allow for a guillotine packing). Ceiling initialisation is always preferred over floor packings, because it delays creation of new levels.

The FCNR algorithm uses the same principle as the BFDH algorithm (Section 2.3), in that a rectangle is packed into the level with minimum residual horizontal space. The residual horizontal ceiling space on each level is computed first, and if none of the rectangles can initialise or be packed on the ceiling, the residual floor space on each level is computed. On the floor, the horizontal space is the distance between the right-hand edge of the last rectangle packed and the right-hand edge of the strip. If, when packing a floor feasible rectangle, the distance between the top edge of the rectangle and the ceiling is insufficient to accommodate any of the unpacked rectangles, then the right-hand edge of such a rectangle forms a *left boundary*. On the ceiling, the horizontal space is the distance between the *left boundary* and the left-hand edge of a rectangle. A new level is created when none of the rectangles can fit onto a ceiling or floor in all existing levels. When the algorithm is applied to our problem instance in Table 1, a packing height of 47 units is obtained, as shown in Fig. 1(h).

2.6. The knapsack algorithm

The *knapsack* (KP01) algorithm [21] was originally developed as a first phase in solving a bin packing problem. Since bin packing is beyond the scope of this paper, the KP01 algorithm is described here in a strip packing context only. In the KP01 algorithm, rectangles are pre-ordered according to non-increasing height. Each level is initialised by packing a rectangle (L_{j^*} say) with greatest height amongst all unpacked rectangles. After initialisation, the knapsack problem,

$$\left.\begin{array}{ll} \text{maximise} & \displaystyle\sum_{i\in U\setminus\{j^*\}} h(L_i)w(L_i)x_i, \\ \text{subject to} & \displaystyle\sum_{i\in U\setminus\{j^*\}} w(L_i)x_i \le W - w(L_{j^*}), \\ & x_i \in \{0,1\} \quad (i \in U\setminus\{j^*\}), \end{array}\right\} \tag{1}$$

is solved for the particular level, where U represents the set of unpacked rectangles at each stage of the packing. Clearly, before any packing takes place $n_1 = n - 1$ since the tallest rectangle has initialised the level. The solution of the knapsack problem identifies those rectangles that should be packed on a particular level (*e.g.* $x_2 = 1, x_3 = 0$ means that rectangle L_2 should be packed and L_3 should not be packed on the particular level in question). The algorithm continues in this manner until all rectangles have been packed. When applied to our example instance in Table 1, a packing height of 52 units is achieved, as shown in Fig. 1(i).

3. Possible variations

Six simple variations on the algorithms described in Sections 2.1–2.3 are proposed in this section.

3.1. Variations of the NFDH algorithm

Two possible variations to the NFDH algorithm, namely the *next-fit decreasing height increasing width* (NFDHIW) algorithm and the *next-fit decreasing height decreasing width* (NFDHDW) algorithm, are proposed. These algorithms are similar to the original algorithm, the only difference being that ties in the sorting order are resolved by additionally sorting according to

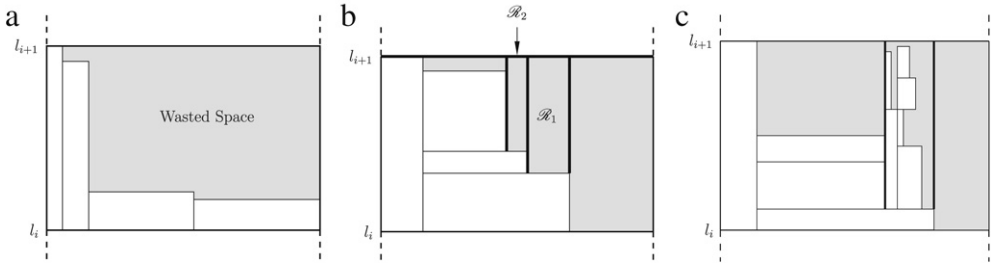

Fig. 2. (a) Motivation for developing the SAS algorithm, (b) Empty rectangular regions left when stacking *wide* rectangles by means of the SAS algorithm and (c) Stacking of narrow rectangle in region $\mathscr{R}_j$.

non-decreasing (resp. non-increasing) width in the NFDHIW (resp. NFDHDW) algorithm. The total height of the packing achieved by the NFDHIW (resp. NFDHDW) algorithm is 53 (resp. 66) units for our example instance in Table 1, as shown in Fig. 1(b) (resp. Fig. 1(c)).

3.2. Variations of the FFDH algorithm

We propose similar, slight variations on the FFDH algorithm, called the *first-fit decreasing height increasing width* (FFDHIW) and *first-fit decreasing height decreasing width* (FFDHDW) algorithms. These heuristics are analogous to the FFDH algorithm—the only difference being that ties in the pre-ordering of rectangles of equal height are additionally resolved according to non-decreasing width and non-increasing width respectively. A packing height of 53 (resp. 52) units is obtained by the FFDHIW (resp. FFDHDW) algorithm when using this procedure for our example instance in Table 1, as illustrated in Fig. 1(e) (resp. Fig. 1(f)).

3.3. Variations of the BFDH algorithm

Similarly proposed modifications to the BFDH algorithm are called the *best-fit decreasing height increasing width* (BFDHIW) and *best-fit decreasing height decreasing width* (BFDHDW) algorithms. The BFDHIW (resp. BFDHDW) algorithm is similar to the BFDH algorithm, except that ties between rectangles of equal height are additionally resolved by ordering them according to non-decreasing width (resp. non-increasing width). When the BFDHIW (resp. BFDHDW) algorithm is applied to our example instance in Table 1, a packing height of 53 (resp. 52) units is achieved, as shown in Fig. 1(e) (resp. Fig. 1(f)).

4. A new level algorithm

It was observed that when the difference in heights of rectangles fitting into one level (see Fig. 2(a)) becomes extreme, the FFDH algorithm performs poorly in relation to an optimal solution. This observation provided motivation for the development of a new algorithm, which we call the *Size Alternating Stack* (SAS) algorithm. In this algorithm, the list $\mathscr{L}$ of n rectangles is partitioned into two sublists $\mathscr{L}_1$ and $\mathscr{L}_2$ consisting of rectangles satisfying $h(L_i) > w(L_i)$ and $h(L_j) \leq w(L_j)$ respectively. The n_1 rectangles in $\mathscr{L}_1$ are called *narrow* rectangles and the n_2 rectangles in $\mathscr{L}_2$ are referred to as *wide* rectangles ($n = n_1 + n_2$). Rectangles in the list $\mathscr{L}_1$ are ordered according to non-increasing height, while rectangles in the list $\mathscr{L}_2$ are ordered according to non-increasing width. Each level of the packing is initialised by comparing the heights of the first rectangle in both lists (*i.e.* the tallest rectangle in $\mathscr{L}_1$ and the widest rectangle in $\mathscr{L}_2$)—packing the rectangle of largest height. The height of this rectangle becomes the height of the next level and a horizontal line is drawn coinciding with the top edge of the rectangle to the right-hand edge of the strip to demarcate the upper boundary of the level.

The main idea in this algorithm is to alternate between the *narrow* and *wide* rectangles while packing from the left to the right on the lower boundary of each level of the strip (*i.e.* if the rectangle initialising a level is from $\mathscr{L}_1$, then we alternate to a rectangle in $\mathscr{L}_2$ and *vice-versa*). Once the list from which to pack has been identified, the rectangles in that particular list are stacked on top of each other, starting from the lower boundary of a level until the upper boundary is reached, or until the vertical space between the upper boundary of the level and the top edge of the top-most rectangle in the stack is insufficient to accommodate any of the unpacked rectangles in that list.

If, when stacking the wide rectangles, it occurs that the widths of subsequent rectangles L_i and L_{i+1} are not equal, an empty rectangular region[3] remains, whose left-hand boundary is the right-hand edge of rectangle L_{i+1}, as shown in Fig. 2(b). The height of each rectangular region extends from the top edge of rectangle L_i to the upper boundary of a level, while the width of each region is given by $w(\mathscr{R}) = w(L_i) - w(L_{i+1})$. These rectangular regions are used to pack the *narrow* rectangles. Whether packing in the region $\mathscr{R}_j$ or through alternation of sizes on the lower boundary of the level, the *narrow* rectangles are stacked by selecting all rectangles whose widths do not exceed the width of the bottom-most narrow rectangle, but also

[3] Such a region is only created when stacking rectangles of unequal widths.

fit height-wise within a level. The stacking of narrow rectangles in $\mathscr{R}_j$ is shown in Fig. 2(c) where the rectangles are stacked to fill the width of $\mathscr{R}_j$ until there is insufficient horizontal space or there are no more *narrow* rectangles to pack.

The algorithm is flexible in the sense that if there is insufficient horizontal space on a level to pack any of the rectangles in the designated list, then the rectangles in the alternative list may be packed, provided that there is sufficient space. A new level is initialised if none of the rectangles in either $\mathscr{L}_1$ or $\mathscr{L}_2$ fit into the horizontal space between the right-hand boundary of the strip and the right-hand edge of the right-most rectangle packed.

The SAS algorithm produces a guillotine packing. The algorithm is given in pseudocode as Algorithm 1. When the algorithm is applied to our example instance in Table 1, the list is first partitioned into lists of wide $(L_2, L_{12}, L_4, L_{11}, L_5, L_{10}, L_1, L_9, L_7)$ and narrow (L_3, L_8, L_6) rectangles. Rectangle L_3 is selected to initialise the first level because its height is greater than that of rectangle L_2. Since rectangle L_3 is from the narrow list, we then alternate to the wide list and stack rectangles L_2 and L_{12}. The region $\mathscr{R}_1$ is created and rectangle L_6 is packed there. To pack on the lower boundary of the rectangle, we alternate to the narrow list where none of the remaining narrow rectangles fit. Since the algorithm is flexible, we move on to the wide list and, in this case, none of the remaining wide rectangles fit, so that a new level is created. Continuing in this manner, a total packing height of 57 units is obtained, as shown in Fig. 1(j).

Algorithm 1 The size alternating stack (SAS) algorithm

Input: The number of rectangles to be packed n, the dimensions of the rectangles $\langle w(L_i), h(L_i)\rangle$ and the strip width W.
Output: The height H of the packing obtained in the strip.

1: Partition the list of rectangles $\mathscr{L} = \mathscr{L}_1 \cup \mathscr{L}_2$ such that $\mathscr{L}_1$ is a list with $h(L_i) > w(L_i)$ for all $1 \le i \le n_1$, while $\mathscr{L}_2$ is a list with $w(L_j) \ge h(L_j)$ for all $1 \le j \le n_2$.
2: Order $\mathscr{L}_1$ according to non-increasing height and order $\mathscr{L}_2$ according to non-increasing width. $j \leftarrow 1,\ i \leftarrow 1,\ \text{level} \leftarrow 1$
3: **while** $n_1 \neq 0$ or $n_2 \neq 0$ **do**
4: compare $h(L_i)$ with $h(L_j)$ and select the rectangle with greatest height. Pack the selected rectangle on the level
5: **if** tallest rectangle is narrow **then**
6: $h(\text{level}+1) \leftarrow h(\text{level}) + h(L_i)$
7: **call** PackWide(w(packedlevel), VerticalSpace)
8: **else** {tallest rectangle is wide}
9: $h(\text{level}+1) \leftarrow h(\text{level}) + h(L_j)$
10: **call** PackNarrow(w(packedlevel), VerticalSpace)
11: **end if**
12: level $\leftarrow$ level $+ 1,\ j \leftarrow 1,\ i \leftarrow 1$
13: **end while**

Procedure 1.1 PackNarrow(w(packedlevel), VerticalSpace)

1: pack first narrow rectangle that fits height-wise and width-wise
2: **while** there is sufficient vertical and horizontal space **do**
3: search $\mathscr{L}_1$ for a rectangle whose width is at most the width of the bottom-most narrow rectangle
4: **if** such a rectangle exists **then**
5: stack the rectangle; remove it from $\mathscr{L}_1$
6: **end if**
7: **end while**

Procedure 1.2 PackWide(w(packedlevel), VerticalSpace)

1: **while** there is sufficient vertical space and $j \le n_2$ **do**
2: **if** rectangle fits height-wise **then**
3: stack rectangle L_j; remove it from $\mathscr{L}_2$
4: **if** rectangles of unequal widths are stacked **then**
5: region $\mathscr{R}$ is created and narrow rectangles are packed in this region
6: **call** PackNarrow (w(packed$\mathscr{R}$), $h(\mathscr{R})$)
7: **end if**
8: **end if**
9: **end while**

5. Benchmark problems

The use of benchmark problems is a standard approach towards the appraisal of new algorithmic procedures, facilitating comparisons between the space and time efficiencies, and solution qualities of such procedures with those of existing ones.

A number of on-line libraries (see, for example [7,15,26,28]) publish benchmark data on the internet for the purposes of testing algorithms designed to solve a large variety of well-documented problems in the operations research literature. However, researchers often test their algorithms on data sets newly created by themselves, but then fail to make these data available when they publish the performance appraisals of their algorithms (see, for example, [24]). Such failures defeat the purpose of benchmark testing, by causing researchers to keep on generating more and more instances of test data instead of reverting to existing data.

This practice of duplication brings with it the disadvantage of even having to implement existing algorithms in order merely to compare the qualities of solutions obtained by them with those of new algorithms, instead of using published results for the existing algorithms and only implementing the new algorithms in such comparisons. Fortunately some researchers [4,5,12,14], have started publishing their work on packing problems along with the test instances they used and/or generated. We follow suit, by making available on the internet all the test data used in our evaluations [27], even though we have not created our own test data, preferring to use established and documented data instead. We nevertheless re–implemented all the existing algorithms cited in this paper, together with the new ones suggested in Sections 3 and 4, because we were interested in observing their time efficiencies in addition to comparing their solution qualities, and because the results of all six previously published algorithms described in Section 2 were not available for certain instances of these benchmark data.

The benchmark data selected for testing the various strip packing heuristics described in Sections 2–4, possess a large variety of combinations of data set and rectangle sizes in order to allow us to determine whether or not certain heuristics have the propensity to perform better on certain types of data. The guillotineable and non-guillotineable[4] data sets described in this section were selected for use in this paper.

5.1. Mumford–Valenzuela benchmark data

Mumford–Valenzuela et al. [24] generated two classes of data sets. The first class is called the class of *Nice* data sets and each data set in this class consists of rectangles of similar sizes and shapes, whilst the second class is referred to as the class of *Path* data sets (short for pathological) and each data set in this class consists of rectangles with significantly varying shapes and sizes. The procedure used to generate these data sets allowed Mumford–Valenzuela et al. [24] to specify the aspect and area ratios of the rectangles. The *Nice* and *Path* data sets have aspect ratios within the ranges $1/4 \leq H/W \leq 4$ and $1/100 \leq H/W \leq 100$ respectively. The maximum ratios of the areas of any two rectangles were set at 7 for the *Nice* data set and at 100 for the *Path* data set. Guillotineable data sets of size n with known optimal solutions were created by performing $n - 1$ guillotine cuts on a 100×100 square. For each class of data sets, the sizes $n = 25, 50, 100, 200, 500$ were selected and these instances are denoted Nice.n and Path.n. Fifty test instances for each problem size within each class were created, except for the case $n = 500$, where only ten test instances were created, resulting in a total of four hundred and twenty data sets.

5.2. Hopper and Turton benchmark data

Hopper and Turton [12] developed test data sets denoted C_i^j, divided into seven categories ($i = 1, \ldots, 7$), each category comprising three test instances ($j = 1, 2, 3$). The number of rectangles in each category ranges from 17 to 197 rectangles. These data sets were generated randomly, maintaining a maximum aspect ratio of 7 and optimal solutions to all test instances are known. The data may be accessed via the on-line libraries [7,15,28].

5.3. Hopper benchmark data

Hopper [13] generated classes of both guillotineable data sets (denoted T_i^j) and non-guillotineable data sets (denoted H_i^j). Each class ($i = 1, \ldots, 7$) comprises five instances ($j = 1, \ldots, 5$) which were cut from a 200×200 square. The guillotineable test problems were generated by repeatedly selecting a random point in a rectangle and performing vertical and horizontal cuts through that point to generate four new rectangles, as depicted in Fig. 3(a). At each iteration (when performing the cuts) the aspect ratio was preserved, failing which a new random point was selected. In the case of the non-guillotineable test problems, two random points were selected within a rectangle, which served as the opposite corner points of a smaller rectangle lying wholly within the larger rectangle, as depicted in Fig. 3(b). The sides of the smaller rectangle were extended up to a point where they intersect the sides of the larger rectangle — thus forming five new rectangles. These data sets are available on-line [28].

5.4. Burke benchmark data

Burke et al. [4] generated thirteen test instances of which only twelve were used here (denoted $B_1, \ldots, B_{12}$), because we only considered data sets containing at most 500 rectangles. These instances were randomly generated by cutting a large rectangle repeatedly, either vertically or horizontally such that two new rectangles were generated after each cut,

[4] Data sets produced without the restriction that guillotine cuts should be possible.

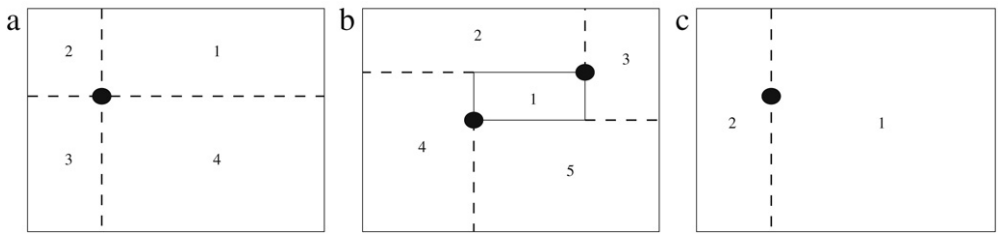

Fig. 3. Generation of guillotine and non-guillotine cuts; (a) guillotine cuts resulting in 4 rectangles, (b) non-guillotine cuts resulting in 5 rectangles and (c) a guillotine cut resulting in 2 rectangles.

as depicted in Fig. 3(c). While ensuring that the dimensions of the rectangles thus generated were not smaller than some specified minimum dimension, the process was carried out until the required number of smaller rectangles was obtained.

5.5. Christofides and Whitlock benchmark data

Some benchmark problems proposed for cutting stock problems were transformed and adapted to strip packing instances by taking the width of the large rectangle from which the smaller rectangles in each test instance were cut as the strip width. The data set generated by Christofides and Whitlock [5] consists of three instances, denoted G_1, G_2, G_3 and were intended for the constrained guillotine cutting problem.[5] The areas of the m smaller rectangles (denoted α_i, $i = 1, \ldots, m$) were generated by sampling from a uniform distribution in the range $[0, 0.25A]$, where A denotes the area of the initial large rectangle. After obtaining the areas of the m rectangles, the height of each rectangle, $h(L_i)$, was obtained by again sampling from a uniform distribution in the range $[0, \alpha_i]$, rounding up to the nearest integer. Finally, the width of each rectangle, $w(L_i)$, was simply computed using the formula $w(L_i) = \lceil \alpha_i / h(L_i) \rceil$. Optimal strip packing solutions for these data sets are unknown and the data are available on-line [26,28].

5.6. Beasley benchmark data

Beasley [1] generated a number of data sets (denoted U_1, U_2, U_3, U_4 and available in [7]) for the unconstrained guillotine cutting problem. Integers were sampled from uniform distributions in the ranges $[H/4, 3H/4]$ and $[W/4, 3W/4]$ for respectively the height $h(L_i)$ and width $w(L_i)$ of each rectangle generated. Beasley [2] also produced twelve non-guillotineable, random problem instances (denoted $V_1, \ldots, V_{12}$), by generating m real numbers r_i ($i = 1, \ldots, m$) from a uniform distribution in the range $(0, HW/4)$. The height, $h(L_i)$, of a rectangle was generated by sampling an integer from the uniform distribution in the range $[1, H]$ and the width was set to $w(L_i) = \lceil r_i / h(L_i) \rceil$. The optimal solutions provided in the original paper are for cutting problems. However, Martello et al. [22] were able to determine optimal solutions to some of these benchmark instances within the context of strip packing, using an exact algorithm. The data may be accessed on-line [26].

6. Comparison of algorithmic results

Experimental results achieved by means of the algorithms considered in Sections 2 and 4 are provided in this section, for the 542 benchmark data sets described in Section 5. The total packing height achieved in each test case, and the frequency with which an algorithm achieves the smallest packing height over all test instances, were used as criteria when comparing and ranking the algorithms—ideally, the best heuristic would be able to attain the smallest strip height in the shortest possible time.

Initially, each original heuristic in Sections 2.1–2.3 is compared with its proposed variations and finally all algorithms are compared in terms of their efficiency and performance. Standard statistical analyses, such as ANalyses Of VAriance (ANOVA) and χ^2–tests were carried out to test for statistically significant differences between the mean strip heights obtained by the different classes of algorithms and the frequencies with which algorithms achieved smallest strip heights. The algorithms were also compared in terms of how close the strip heights obtained were to an optimal solution (if known) by using the performance ratio,

$$PR(A) = A(\mathscr{L})/OPT(\mathscr{L}), \tag{2}$$

where $A(\mathscr{L})$ is the strip height obtained by algorithm A when applied to the benchmark instance $\mathscr{L}$, and where $OPT(\mathscr{L})$ is the optimal height associated with the data set $\mathscr{L}$.

6.1. Comparison of algorithms in the next fit class

The relative performances of the NDFH, NFDHIW and NFDHDW algorithms were assessed as the ability of each algorithm to obtain the smallest strip height. The mean strip heights obtained by these algorithms over the 542 benchmark data

[5] The problem of cutting a large rectangle of fixed dimensions (H, W) into m smaller rectangles of specified areas.

Table 2
Summary of results from an analysis of variance when comparing mean strip heights obtained by algorithms in the classes of *next fit*, *first fit* and *best fit* heuristics.

NFDH	NFDHIW	NFDHDW	FFDH	FFDHIW	FFDHDW	BFDH	BFDHIW	BFDHDW	F_{value}	$F_{critical}$
169.542	169.488	169.708							0.00016	3.00127
			161.570	161.875	161.399				0.00104	3.00127
						161.360	161.607	161.240	0.00063	3.00127

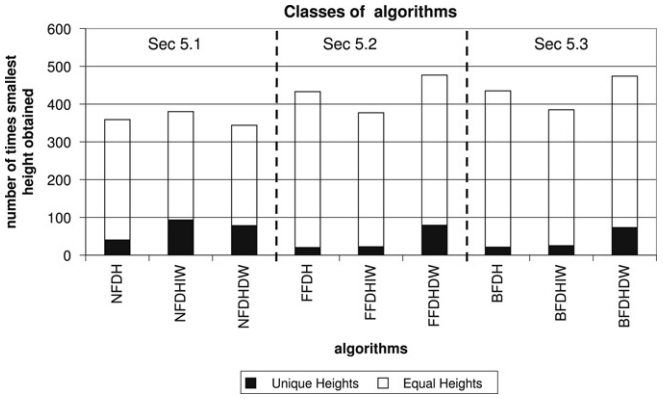

Fig. 4. Frequencies with which algorithms achieved the smallest strip height with respect to the 542 benchmark data sets described in Section 5. The results for each class of algorithms were obtained separately over the 542 benchmark data.

Table 3
Summary of results from the chi-squared test for frequencies with which algorithms obtained the smallest packing height with respect to the 542 benchmark data sets described in Section 5. The bold faced frequencies represent the highest frequencies within a class, while bold faced χ^2_{df} and $\chi_{critical}$ values denote no significant difference between frequencies.

NFDH	NFDHIW	NFDHDW	FFDH	FFDHIW	FFDHDW	BFDH	BFDHIW	BFDHDW	$\chi^2_{df}(0.05)$	$\chi_{critical}$
359	**380**	344							**1.812**	**5.990**
			433	377	**477**				11.711	5.990
						435	385	**474**	9.229	5.990

sets are shown in Table 2, together with two values, F_{value} and $F_{critical}$, in each case. Here, F_{value} represents the fraction of variance between the packing heights obtained by the algorithms and the variance of strip heights within each algorithm, while $F_{critical}$ is the associated test statistic obtained from an F-distribution table. The results of the analysis of variance in the *next fit* class comparison show that there is no significant difference between the mean strip heights obtained by the algorithms in the class at a 5% level of significance. This result is not surprising, given the similar natures of the algorithms in question.

The next step was to assess how often each algorithm obtained the smallest packing height. The results are shown graphically in Fig. 4, in which the first three bars represent the frequencies for the *next fit* class of algorithms. The heights of the unshaded bars in the figure represent the number of times each algorithm obtained the smallest packing height, and the heights of the shaded bars represent the number of times each algorithm was the only procedure to obtain the smallest packing height (*i.e.* the number of times the algorithm obtained the smallest packing height uniquely). The NFDHIW algorithm was able to obtain the smallest strip height more often than the other two algorithms, as may be seen in Fig. 4.

We were interested in establishing statistically whether there were significant differences between these frequencies. The chi-squared test at a 5% level of significance was employed in this regard, and the results are shown in row 1 of Table 3. Because $\chi_{critical} > \chi^2_{df}(0.05)$, we conclude that statistically there appears to be no difference between the frequencies. Hence, no algorithm in the *next fit* class is statistically superior to another in terms of the frequency with which it achieved the best results, at a 5% level of significance.

Because the three algorithms in the *next fit* class can never perform better than their counterparts in the *first fit* and *best fit* classes, these algorithms were excluded from all further comparisons and analyses.

6.2. Comparison of algorithms in the first fit class

Similar to the analysis carried out in Section 6.1, a comparison of the relative performances of the FFDH, FFDHIW and FFDHDW algorithms was also based on the mean strip heights obtained by each algorithm. An analysis of variance indicated that there is no significant difference between the mean strip heights obtained by the three algorithms over all 542 benchmark data sets at a 5% significance level. This may be seen from row 2 of Table 2 in which the computed F_{value}

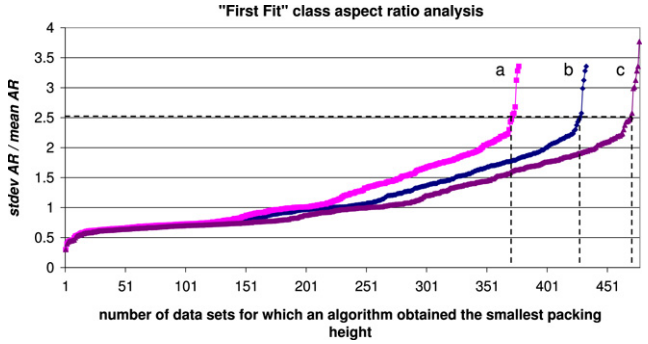

Fig. 5. Analysis of the aspect ratio variation of data sets for the *first fit* class of algorithms: a — FFDHIW, b — FFDH and c — FFDHDW.

(0.00104) is less than F_{critical} (3.00127). Hence, in terms of the overall quality of the solution, there is no statistical difference between the algorithms in the *first fit* class—all of the algorithms are expected to perform approximately equally effectively.

From row 2 of Table 3 it is evident that the FFDHDW algorithm outperformed the other two algorithms in terms of the frequency with which it obtained the smallest packing height. The reason for this observation is that in the FFDHDW algorithm, wider rectangles are packed first among rectangles of equal height, and since existing levels are always searched for sufficient space, the smaller rectangles may fit into any of the levels. This reduces the necessity of having to create new levels, which in turn results in smaller strip heights.

The chi-squared test was used to determine whether there are any statistically significant differences between the frequencies of obtaining the smallest packing height by any of the *first fit* class of algorithms. In row 2 of Table 3, $\chi_2^2 > \chi_{\text{critical}}$ at a 5% level of significance with two degrees of freedom, implying that there are significant differences between the frequencies. In terms of this outcome, the FFDHDW algorithm takes first preference, because it achieves the highest frequency. The chi-squared test was carried out separately using a Yates correction [30] (because there was now only one degree of freedom) to ascertain whether the frequencies obtained by the FFDH and FFDHIW algorithms were statistically distinguishable. The results of this test indicate that they are indeed distinguishable with $(\chi_1^2 = 5.033) > (\chi_{\text{critical}} = 3.84)$, hence the FFDH algorithm ranks second, and the FFDHIW algorithm third, within the first fit class of algorithms at a 5% level of significance.

The number of data sets for which each algorithm was able to find the smallest packing height is shown against the ratio of the standard deviations of the aspect ratio of the data set (*stdev AR*) with respect to the mean aspect ratio (*mean AR*) in Fig. 5. For a fixed ratio of *stdev AR*/*mean AR* = 2.5, for example, as indicated in the figure by the horizontal dashed line, the number of data sets with ratio less than or equal to 2.5 for which the FFDHIW, FFDH and FFDHDW algorithms were able to obtain the smallest strip height are given by 373, 429 and 471 respectively (indicated by the vertical dashed lines). For data sets with little variation in the rectangle aspect ratios, the algorithms in the *first fit* class seem indistinguishable in terms of the frequency with which they are able to obtain the smallest strip height, as may be seen in Fig. 5. However, as the variation in the rectangle aspect ratios increases, the algorithms become distinguishable in terms of the above mentioned frequency, as may be seen in the figure.

An investigation into the aspect ratios (*AR*) of the data sets was carried out to determine for what ratio of *stdev AR*/*mean AR* in each data set, each algorithm was able to obtain the smallest height. This was done by considering the ratios of *stdev AR*/*mean AR* for which each algorithm was able to attain the smallest strip height, and performing a chi-squared test for a range of values of the ratio *stdev AR*/*mean AR* up to a point where $\chi_{\text{df}}^2 \approx \chi_{\text{critical}}$ at a 5% significance level. Such a ratio becomes a threshold value in the sense that, for any data set with *stdev AR*/*mean AR* ratio beyond the threshold, the FFDHDW algorithm is the preferred choice (because for a fixed standard deviation above this threshold, it obtains the smallest height in a larger number of data sets than the other two algorithms, as may be seen in Fig. 5), while for ratios below the threshold, any one of the remaining two algorithms may be used. In this class, the threshold value is 0.776.

6.3. Comparison of algorithms in the best fit class

The results of the analysis of variance, when comparing the mean strip heights obtained by the algorithms in the *best fit* class, indicate that there is no difference between the mean strip heights achieved by the algorithms at a 5% level of significance. This may be seen from row 3 of Table 2 where $F_{\text{value}}(0.00063) < F_{\text{critical}}(3.00127)$. Hence, in terms of solution quality, no algorithm is superior with respect to another.

Based on the results of the chi-squared test (row 3 of Table 3), there are significant differences between the frequencies at which the smallest packing height was obtained by the three algorithms, because $(\chi_{\text{df}}^2 = 9.229) > (\chi_{\text{critical}} = 5.990)$. As shown in Fig. 4, the BFDHDW algorithm obtained the smallest strip height more often than the other two algorithms in this class. The chi-squared test with a Yates correction was carried out separately to determine whether the frequencies obtained by the BFDH and BFDHIW algorithms were statistically further distinguishable, and the results indicate that they

Table 4
Summary of results from the analysis of variance and the chi-squared test.

BFDH	BFDHDW	BFDHIW	FFDH	FFDHDW	FFDHIW	KP01	FCNR	SAS	SF	F_{value}	F_{critical}
161.360	161.240	161.607	161.570	161.399	161.875	164.292	148.872	157.046	173.289	0.610	1.882
BFDH	BFDHDW	BFDHIW	FFDH	FFDHDW	FFDHIW	KP01	FCNR	SAS	SF	$\chi^2_{\text{df}}(0.05)$	χ_{critical}
33	40	24	32	39	22	30	399	136	8	1661.05	16.92

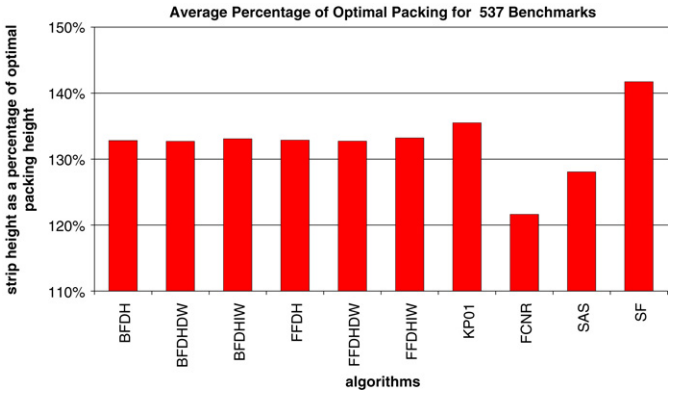

Fig. 6. Overall comparison between the packing heights obtained by each algorithm and the optimal solutions.

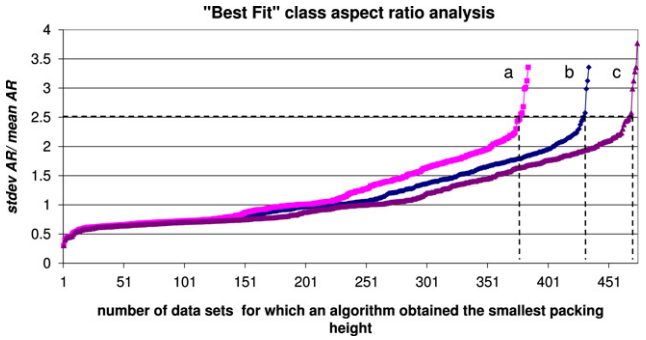

Fig. 7. Analysis of the aspect ratio variation of data sets for the *Best Fit* class of algorithms: a — BFDHIW, b — BFDH and c — BFDHDW.

are distinguishable ($\chi_1^2 = 3.917$) > ($\chi_{\text{critical}} = 3.840$). The BFDH algorithm therefore ranks second, and the BFDHIW third, within the best fit class of algorithms.

In Fig. 7, for a fixed ratio of $stdev\ AR/mean\ AR = 2.5$, for example, indicated by the horizontal dashed line, the number of data sets with ratios less than or equal to 2.5 for which the BFDHIW, BFDH and BFDHDW algorithms were able to obtain the smallest strip height are given by 378, 430 and 468 respectively (indicated by the vertical dashed lines). In the *best fit* class, a threshold value of 0.79 was obtained. The BFDHDW algorithm is the preferred choice when dealing with a data set whose *stdevAR/meanAR* ratio is above the threshold value, as may be seen in Fig. 7.

6.4. Comparison of the ten heuristics

The average performance ratios (PR) expressed as percentages for all algorithms are shown in Fig. 6. This particular analysis was performed on only 537 benchmark instances, because optimal solutions are not known for the remaining five data sets.

The SF, SAS, KP01 and FCNR algorithms are compared to the six algorithms from the *best fit* and *first fit* classes in this section. As shown in Fig. 6, the FCNR algorithm was, on average, within 21% of the optimal solution over all the 537 data sets. It is followed closely by the new SAS algorithm which yields solutions within 28% of the optimal solution. The SF algorithm differs from the other algorithms in that it allows the *wide* rectangles already packed to be shifted around, but the results in Fig. 6 indicate that this advantage over the other algorithms does not render the algorithm superior with respect to the other procedures—in fact, it exhibited the lowest frequency in obtaining the smallest packing height, as shown in Fig. 8.

The first section of Table 4 represents the results from the analysis of variance, indicating that there is no difference between the mean strip heights obtained by the ten algorithms at a 5% level of significance. The second section contains the

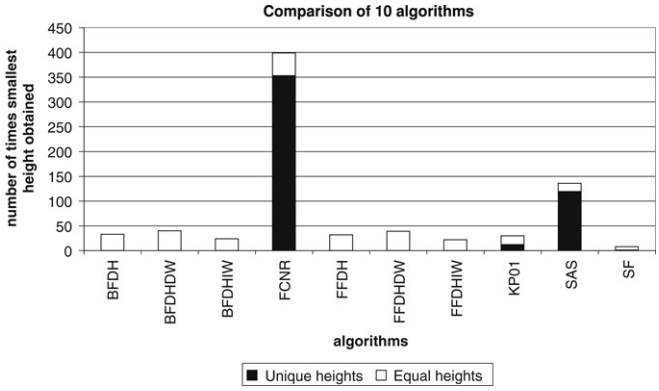

Fig. 8. The number of times each of the 10 algorithms obtained the smallest strip height.

Table 5
Summary of mean packing heights obtained by offline algorithms over the 542 data sets described in Section 6.4.

Author	Data sets	# of Rectangles	OPT	FFDH	FFDHDW	FFDHIW	BFDH	BFDHDW	BFDHIW	SF	KP01	SAS	FCNR
Beasley [1,2]													
	U_1	10	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016	1016
	U_2	20	–	1349	1349	1349	1349	1349	1349	1382	1347	1499	1349
	U_3	30	1803	1873	1873	1873	1810	1810	1810	2446	1899	2077	1810
	U_4	50	–	3216	3216	3216	3216	3216	3216	4043	3159	3396	3216
	V_1	10	23	25	25	25	25	25	25	25	25	27	25
	V_2	17	30	33	33	33	33	33	33	33	36	33	33
	V_3	21	28	34	31	34	34	34	34	34	35	31	34
	V_4	7	20	23	23	23	23	23	23	23	23	23	23
	V_5	14	36	46	37	46	46	37	46	46	37	37	46
	V_6	15	31	38	38	40	38	38	40	38	38	35	36
	V_7	8	20	21	21	21	21	21	21	22	21	20	21
	V_8	13	33	38	44	38	38	44	38	44	38	38	38
	V_9	18	–	65	65	66	65	65	65	65	65	60	64
	V_{10}	13	80	85	85	85	85	85	85	85	93	85	85
	V_{11}	15	52	69	69	69	69	69	69	69	69	75	63
	V_{12}	22	87	104	104	104	104	104	104	104	102	91	96
Burke et al. [4]													
	B_1	10	40	46	46	46	46	46	46	48	46	60	46
	B_2	20	50	65	65	65	65	65	67	67	61	65	61
	B_3	30	50	68	68	68	68	68	68	76	68	63	62
	B_4	40	80	126	126	126	126	126	126	140	126	103	93
	B_5	50	100	119	119	125	119	119	125	120	122	115	111
	B_6	60	100	109	109	110	109	109	110	131	109	110	105
	B_7	70	100	160	160	160	160	160	160	161	163	120	124
	B_8	80	80	108	108	108	108	108	108	115	115	104	92
	B_9	100	150	181	181	181	177	177	177	190	177	177	162
	B_{10}	200	150	210	191	212	190	191	192	217	192	158	182
	B_{11}	300	150	169	169	171	169	169	171	170	168	159	155
	B_{12}	500	300	372	372	372	372	372	372	375	372	341	343
Christofides [5]													
	G_1	16	23	28	28	28	28	28	28	28	28	25	25
	G_2	23	–	83	83	83	83	83	83	83	84	75	73
	G_3	62	–	744	728	744	744	728	744	796	728	744	744

results from the chi-squared test on the number of times each algorithm obtained the smallest strip height, and the results indicate that there is a significant difference between these frequencies. Based on the results from the frequency analysis, the FCNR algorithm yields superior solutions, followed by the newly proposed SAS algorithm. This was an expected result, because both these algorithms attempt to utilise each level fully, thereby often leading to reduced total packing heights (the FCNR algorithm by utilising both the floor and ceiling of a level, and the SAS algorithm by stacking rectangles vertically within a level)—hence these algorithms have an advantage over the other (classical) level algorithms, leading to performance superiority. A summary of the packing heights obtained by each heuristic over the 542 benchmark instances is shown in Tables 5 and 6.

Table 6
Summary of mean packing heights obtained by offline algorithms over the 542 data sets described in Section 6.4.

Author	Data sets	# of Rectangles	OPT	FFDH	FFDHDW	FFDHIW	BFDH	BFDHDW	BFDHIW	SF	KP01	SAS	FCNR
Hopper et al. [12,13]													
	C1	16/17	20	27.3	27.3	28.0	27.3	27.3	27.3	29.3	27.7	25.7	22.0
	C2	25	15	18.0	17.3	18.3	18.0	17.3	18.3	19.0	17.0	19.0	17.0
	C3	28/29	30	38.0	38.0	38.3	38.0	38.0	38.3	39.7	38.3	36.3	35.3
	C4	49	60	77.0	76.0	77.7	76.3	76.0	76.3	81.0	76.3	68.0	69.0
	C5	72/73	90	104.7	104.7	105.0	104.7	104.7	105.0	105.7	104.7	102.3	98.3
	C6	97	120	140.7	140.0	141.7	140.7	140.0	141.0	144.3	142.0	135.3	128.7
	C7	196/197	240	272.7	272.7	273.3	272.3	272.3	273.3	277.3	273.7	266.0	252.7
	H1	17	200	275	275.0	275	271.8	271.8	271.8	276.4	278.2	259.4	242.6
	H2	25	200	266.2	266.2	266.2	266.2	266.2	266.2	282.6	269.6	269.4	244.8
	H3	29	200	273.6	273.6	273.6	273.6	273.6	273.6	281.8	273.2	259.6	246
	H4	49	200	257.2	257.4	257.4	257.2	257.4	257.4	261.4	262.	244.8	237.6
	H5	73	200	256.6	256.4	256.6	256.4	256.4	256.4	264	260.8	238	227.8
	H6	97	200	256.4	256.2	256.4	256.2	256.2	256.4	261.4	259.2	235.4	230
	H7	197	200	248.2	248.0	248.6	248.2	248	248.6	254.2	250.8	229.4	221.2
	T1	17	200	293.0	293.0	293.0	293.0	293.0	293.0	297.2	293.0	282.4	274.4
	T2	25	200	281.6	281.6	281.6	281.6	281.6	281.6	289.6	287.8	268.8	263.2
	T3	29	200	281.4	281.6	281.4	281.2	281.4	281.2	287	284	269.2	251.8
	T4	49	200	255	255.2	255.4	255	255.2	255.4	260	258.8	247.2	235
	T5	73	200	254.2	253.8	254.2	253.8	253.8	253.8	260.8	257.6	234.8	228.2
	T6	97	200	263.4	263.4	263.4	263.4	263.2	263.4	265.6	266.8	233.6	233.2
	T7	199	200	254.8	254.6	255.2	254.8	254.6	255.2	262.4	255.8	224.6	226
Mumford et al. [24]													
	nice.25	25	100	130.6	130.6	130.7	130.6	130.6	130.7	138.3	132.3	131.1	123.1
	nice.50	50	100	122.2	121.9	122.1	122.2	121.9	122.1	129.1	122.2	125.4	118.1
	nice.100	100	100	117.8	117.7	117.6	117.8	117.7	117.6	122.4	117.8	120.0	113.2
	nice.200	200	100	113.2	113.1	113.3	113.2	113.1	113.3	117.2	112.9	116.1	110.3
	nice.500	500	100	108.2	108.2	108.4	108.2	108.2	108.4	111.8	108.2	112.8	106.5
	path.25	25	100	147.8	147.8	148.1	147.8	147.8	148.0	169.2	153.9	149.0	137.5
	path.50	50	100	149.3	149.2	149.6	149.3	149.2	149.6	168.7	158.7	138.7	137.2
	path.100	100	100	149.6	149.6	149.7	149.7	149.6	149.7	161.6	154.5	131.6	140.2
	path.200	200	100	147.8	147.8	147.8	147.8	147.8	147.8	153.0	150.7	125.3	140.8
	path.500	500	100	142.1	142.0	142.1	142.1	142.0	142.1	144.5	143.0	119.8	137.7

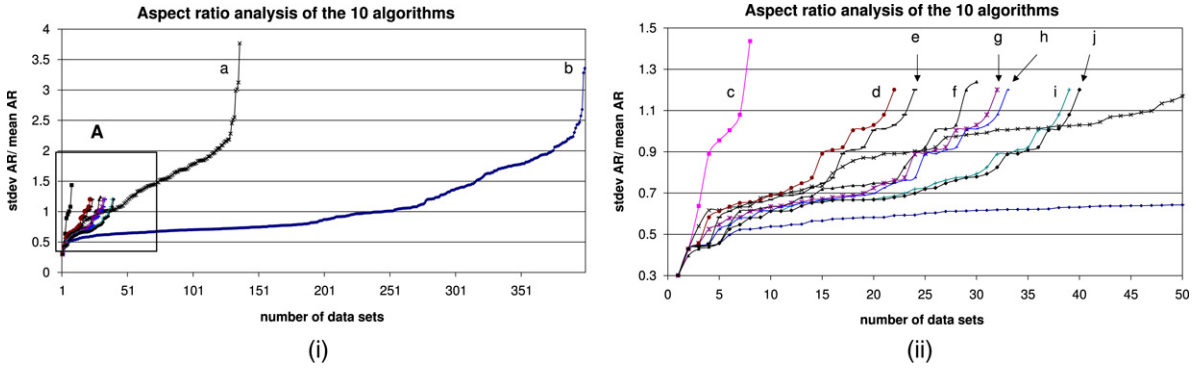

Fig. 9. Analysis of the aspect ratio variation of data sets for 10 algorithms. Subfigure (i): a — SAS, b — FCNR. Subfigure (ii): Enlargement of rectangular region **A** in subfigure (i): c — SF, d — FFDHIW, e — BFDHIW, f — KP01, g — FFDH, h — BFDH, i — FFDHDW and j — BFDHIW.

An analysis of the set of 10 algorithms with respect to the aspect ratios of the data sets is shown in Fig. 9 (i) and (ii)—the latter figure is an enlargement of the section indicated by a rectangular frame **A** in the former figure. A threshold value of 0.564 was obtained for this set of algorithms in the sense that the FCNR algorithm outperforms the other nine algorithms for *stdev AR/mean AR* ratios above this value. However, for some applications, the FCNR algorithm might not be suitable (e.g. where packing on the ceiling might not be feasible). Hence an analysis of the remaining nine algorithms with respect to aspect ratios was carried out. The results of the analyses are shown in Fig. 10. For this set of algorithms, a threshold value of 0.592 was obtained, indicating that the newly proposed SAS algorithm is the preferred choice for data sets whose *stdev AR/mean AR* ratio exceeds the threshold value.

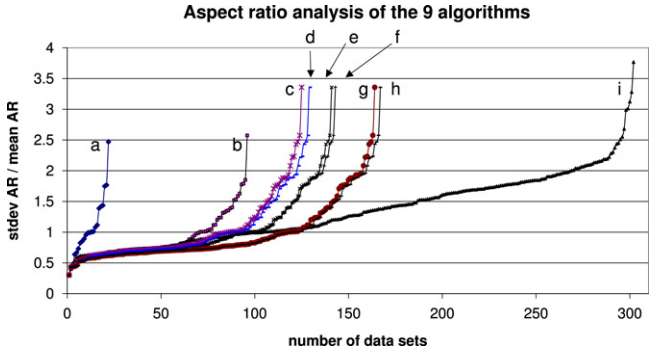

Fig. 10. Analysis of the aspect ratio variation of data sets for 9 algorithms: a — SF, b — KP01, c — FFDHIW, d — BFDHIW, e — FFDH, f — BFDH, g — FFDHDW, h — BFDHIW and i — SAS.

7. Final remarks

In this paper, we implemented a number of level heuristics from the literature and proposed possible improvements to some of these algorithms. We also developed a new heuristic (called the SAS algorithm). A total of ten out of thirteen algorithms were compared in terms of their solution qualities and their ability to obtain the smallest strip height (since three of the algorithms are always outperformed and were not included in the general comparison). The results of the analyses of variance indicate that, statistically, there is no difference between the mean strip heights obtained by the algorithms at a 5% level of significance (*i.e.* in terms of the solutions qualities that they produce). However, there is a considerable difference in the number of times the various algorithms obtained the smallest strip height at a 5% level of significance when applying the algorithms to the benchmark data. Although the differences in mean strip heights are not statistically significant, a small improvement in the total packing height may lead to significant cost savings in industrial applications when applying one algorithm instead of another, thereby prompting an interest in the relative frequencies at which algorithms achieve smallest packing heights.

Although the algorithms were not explicitly compared in terms of time efficiencies, it is interesting to mention that all the algorithms were able to produce a solution in less than one second, with the exception of the KP01 algorithm which took more than 3 hours[6] for large data sets (with 500 rectangles) on a 2.00 GHz processor with 224 MB of RAM.

Acknowledgments

Work towards this paper was supported financially by the Third World Organisation for Women in Science (TWOWS), by the South African National Research Foundation (Gun 2072815) and by the Harry Crossley Foundation. We are grateful to Dr. Werner Gründlingh and Mr. John Part for typesetting assistance and advice with respect to programming implementations, respectively. The two anonymous referees are also thanked for their valuable suggestions with respect to improvement of the paper presentation.

References

[1] J.E. Beasley, Algorithms for unconstrained two–dimensional guillotine cutting, Journal of the Operational Research Society 36 (4) (1985) 297–306.
[2] J.E. Beasley, An exact two–dimensional non-guillotine cutting tree search procedure, Operations Research 33 (1) (1985) 49–64.
[3] A. Bortfeldt, A genetic algorithm for the two dimensional strip packing problem with rectangular pieces, European Journal of Operational Research 172 (2006) 814–837.
[4] E.K. Burke, G. Kendall, G. Whitwell, A new placement heuristic for the orthogonal stock-cutting problem, Operations Research 52 (4) (2004) 655–671.
[5] N. Christofides, C. Whitlock, An algorithm for two–dimensional cutting problems, Operations Research 25 (1) (1977) 31–44.
[6] E.G. Coffman Jr., D.S. Garey, R.E. Tarjan, Performance bounds for level oriented two-dimensional packing algorithms, SIAM Journal on Computing 9 (4) (1980) 808–826.
[7] DEIS (Dipartmento di Elettronica, Informatica e Sistemistica Operations Research Group), [Online], [cited 2005, April 22], Available from: http://www.or.deis.unibo.it/research_pages/ORinstances/ORinstances.htm.
[8] K.A. Dowsland, W.B. Dowsland, Packing problems, European Journal of Operational Research 56 (1992) 2–14.
[9] J. Egeblad, Heuristics for packing problems, [Online], [cited 2008, May 22], Available from: http://www.diku.dk/undervisning/2005e/426/HeuristicPackings.pdf, 2005.
[10] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[11] M. Gradisar, G. Resinovic, M. Kljajic, Evaluation of algorithms for one-dimensional cutting, Computers and Operations Research 29 (2002) 1207–1220.
[12] E. Hopper, B.C.H. Turton, An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem, European Journal of Operational Research 128 (1) (2001) 34–57.
[13] E. Hopper, B.C.H. Turton, Problem generators for rectangular packing problems, Studia Informatica Universalis 2 (1) (2002) 123–136.

[6] The Knapsack was solved to optimality. However, to reduce the execution time, one may utilize an interim solution by imposing stopping criteria such as the maximum number of iterations or maximum time allowed.

[14] E. Hopper, B.C.H. Turton, A review of the application of meta-heuristic algorithms to 2D strip packing problems, Artificial Intelligence Review 16 (4) (2001) 257–300.
[15] S. Imahori, Cutting and packing, [Online], [cited 2005, March 11], Available from: http://www.simplex.t.u-tokyo.ac.jp/~imahori/packing/instance.html.
[16] C. Imreh, Online strip packing with modifiable boxes, Operations Research Letters 29 (2001) 79–85.
[17] C. Kenyon, E. Remila, A near optimal solution to a two-dimensional cutting stock problem, Mathematics of Operations Research 25 (2000) 645–656.
[18] A. Lodi, S. Martello, M. Monaci, Two-dimensional packing problems: A survey, European Journal of Operational Research 141 (2002) 241–252.
[19] A. Lodi, S. Martello, D. Vigo, Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem, in: S. Voss, S. Martello, I.H. Osman, C. Roucairol (Eds.), in: Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Boston, 1998, pp. 125–139.
[20] A. Lodi, S. Martello, D. Vigo, Recent advances on two-dimensional bin packing problems, Discrete Applied Mathematics 123 (2002) 379–396.
[21] A. Lodi, S. Martello, D. Vigo, Heuristics and meta-heuristic approaches for a class of two-dimensional bin packing problems, INFORMS Journal on Computing 11 (4) (1999) 345–357.
[22] S. Martello, M. Monaci, D. Vigo, An exact approach to the strip-packing problem, INFORMS Journal on Computing 15 (3) (2003) 310–319.
[23] Microsoft visual basic developer center, [Online], [cited 2004, January 05], Available from http://msdn.microsoft.com/vbasic.
[24] C. Mumford–Valenzuela, P.Y. Wang, J. Vick, Heuristic for large strip packing problems with guillotine patterns: An empirical study, in: Proc. 4th Metaheuristics Int. Conference, 2001, pp. 417–421.
[25] K.G. Murty, Operations Research: Deterministic Optimisation Models, Prentice Hall, New Jersey, 1995.
[26] OR-Library, [Online], [cited 2005, February 25], Available from: http://mscmga.ms.ic.ac.uk/info.html.
[27] Repositories (Department of Applied Mathematics), [Online], [cited 2005, November 22], Available from: http://dip.sun.ac.za/~vuuren/repositories/strippacking.htm.
[28] SICUP (Special interest group in cutting and packing), [Online], [cited 2005, March 18], Available from: http://www.apdio.pt/sicup.
[29] D. Sleator, A 2.5 times optimal algorithm for packing in two-dimensions, Information Processing Letters 10 (1) (1980) 37–40.
[30] TRESS (The Really Easy Statistics Site), [Online], [cited 2005, November 28], Available from: http://helios.bto.ed.ac.uk/bto/statistics/tress9.html#Chi-squared%20test.
[31] G. Wäscher, H. Haußner, H. Schumann, An improved typology of cutting and packing problems, European Journal of Operational Research, Article in Press, [Online], [cited 2006, October 12], Available from: http://www.sciencedirect.com, 2006.