Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.

This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

Author's personal copy

European Journal of Operational Research 203 (2010) 306–315

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

New and improved level heuristics for the rectangular strip packing and variable-sized bin packing problems

Frank G. Ortmann, Nthabiseng Ntene, Jan H. van Vuuren *

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

ARTICLE INFO

Article history:
Received 29 January 2008
Accepted 28 July 2009
Available online 3 August 2009

Keywords:
Bin packing
Strip packing

ABSTRACT

We consider two types of orthogonal, oriented, rectangular, two-dimensional packing problems. The first is the strip packing problem, for which four new and improved level-packing algorithms are presented. Two of these algorithms guarantee a packing that may be disentangled by guillotine cuts. These are combined with a two-stage heuristic designed to find a solution to the variable-sized bin packing problem, where the aim is to pack all items into bins so as to minimise the packing area. This heuristic packs the levels of a solution to the strip packing problem into large bins and then attempts to repack the items in those bins into smaller bins in order to reduce wasted space. The results of the algorithms are compared to those of seven level-packing heuristics from the literature by means of a large number of strip-packing benchmark instances. It is found that the new algorithms are an improvement over known level-packing heuristics for the strip packing problem. The advancements made by the new and improved algorithms are limited in terms of utilised space when applied to the variable-sized bin packing problem. However, they do provide results faster than many existing algorithms.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

Many types of packing and cutting problems have been studied since the early work by Kantorovich [48] in 1939, Eisemann [25] in 1957, and Gilmore and Gomory [30–32] in the 1960s. Papers on the typology of packing and cutting problems, such as those by Dyckhoff [24] and Wäscher et al. [69], illustrate the vast number of problems that exist in the literature, while surveys, such as those by Sweeney and Paternoster [64], and Coffman et al. [19], show the large volume of work published on these problems. One common aspect is that, typically, small items have to be packed into, or cut from, one or more large objects (often called bins). However, there are many variations to the basic packing problem and each problem variation may have solution procedures designed specifically for it. For example, problems may be one, two, three or N-dimensional (where $N > 3$) in nature, with the aim of maximising the value of items packed, or minimising wasted space. Items and objects may be regular or irregular, and may all have the same shape, be weakly or strongly heterogeneous,[1] or there may be groups of identical items.

Recent exact approaches toward solving two-dimensional packing problems have been developed by Lodi et al. [55] for the strip (open-ended bin) and single-sized bin packing problems, by Martello et al. [56] for the strip packing problem, by Pisinger and Sigurd [60] for the variable-sized bin packing problem with variable costs, and by Clautiaux et al. [18] and Pisinger and Sigurd [61] for the single-sized bin packing problem. Hifi and Zissimopoulos [37] improved the exact algorithm for the constrained two-dimensional cutting problem by Christofides and Whitlock [16], while Cui [22] developed an exact algorithm for the constrained two-dimensional cutting problem. Recently Cui et al. [23] also developed a recursive algorithm incorporating branch-and-bound techniques to solve the strip packing problem where guillotine cuts are required and rotations are allowed.

However, packing problems are $\mathcal{NP}$-hard with the result that much research is focussed on algorithms that sacrifice solution quality in order to solve large problems within reasonable time. One class of such algorithms is metaheuristics. Recent applications of genetic algorithms to packing problems include those by Liu and Teng [49], Hopper and Turton [38–40], Valenzuela and Wang [65], Bortfeldt [12], Gonçalves and Resende [35], and Burke et al. [14]. Simulated annealing methods have recently been employed by authors such as Faina [27], Hopper and Turton [38,40], Beisiegel et al. [8] and Burke et al. [14]. Another popular metaheuristic used for two-dimensional packing problems is tabu search, employed by amongst others Lodi et al. [52,53], Alvarez-Valdes et al. [1,2,4], Pureza and Morabito [62] and Burke et al. [14]. Other search procedures include implementations of GRASP algorithms for the constrained two-dimensional non-guillotine cutting problem and the strip packing problem by Alvarez-Valdes et al. [3,5] and the algorithm by Benati [10] for a cutting stock problem on a strip.

* Corresponding author. Fax: +27 21 808 3406.
E-mail address: vuuren@sun.ac.za (J.H. van Vuuren).

[1] See Wäscher et al. [69] for a detailed description of the terminology.

0377-2217/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2009.07.024

Author's personal copy

A number of heuristics have been developed for the two-dimensional strip and bin packing problems. They are typically used to find solutions very quickly, often at considerable sacrifice with respect to solution quality. They may also be used to find upper bounds on packing heights or bin utilisation [56], find initial feasible solutions [12] or to decode metaheuristics (see [38–40]). Heuristics yield solutions that may be categorised as either level, pseudolevel or plane packings. Level algorithms (see [20,21,53,56]) pack items into levels such that the bottom edges of all items are on the floor of their respective levels (sometimes called two-stage packing). Pseudolevel algorithms also pack items into levels, but without the constraint of floor packing (see [12,51,53,58,59]). Plane-packing algorithms may pack items anywhere in the plane defined by the boundaries of the strip or bin (see [6,7,13–15,20,21,33,34,38–40,43,49,53,63]).

In this paper, we present heuristics for two types of two-dimensional packing problems (see the papers by Lodi et al. [50,54] for recent surveys on two-dimensional packing problems). The first is the problem where a fixed number of rectangular items have to be packed into a single semi-rectangular object of fixed width and infinite height (called a strip). Dyckhoff [24] characterises this as a 2/V/O/* problem, while Wäscher et al. [69] call it an Open Dimension Problem (ODP). The second problem is that of packing the rectangular items into strongly or weakly heterogeneous rectangular bins. This problem is characterised as a 2/V/D/* or 2/V/G/* problem (the "G" was used by Gradišar et al. [36] to indicate the case where there are only few groups of identical bins). This class of problems includes the Multiple Bin Size Bin Packing Problem (MBSBPP) and the Residual Bin Packing Problem (RBPP), as described by Wäscher et al. [69].

We enforce certain restrictions on the type of packing that may occur. All items must be packed orthogonally into the bin or strip (*i.e.* the edges of the items must be parallel to the edges of the bin or strip), the items may not overlap and no rotations may occur during the packing process (*i.e.* an oriented packing is sought). Furthermore, sometimes a guillotineable packing is sought (in such cases it should be possible to disentangle the packing by means of cuts that pass from one edge of the bin/strip to the opposite edge, without passing though another item).

We assume that the dimensions of all items and objects are known prior to packing; such packing problems are known as off-line problems. For on-line problems, the dimensions of an item only become known once the previous item has been packed.

2. Known heuristics

This section contains a brief summary of the literature on level-packing heuristics for the strip packing problem and heuristics for bin packing problems.

2.1. Strip packing

A strip is an open-ended bin whose width is fixed, but whose height is infinite. The distance from the bottom of the strip to the top edge of the highest item packed in the strip is called the *packing height*. The objective in the strip packing problem is to pack all items in a given list so that the resulting packing height is as small as possible.

In order to test the four new and improved strip packing algorithms introduced in this paper, the efficiencies of the algorithms are compared with those of seven published level-oriented strip packing algorithms. These algorithms partition the strip using horizontal lines creating levels. A level is initialised above the topmost level if items no longer fit on lower levels. The first of these algorithms, the *first-fit decreasing height* (FFDH) algorithm, was developed by Coffman et al. [20] in 1980. The *best-fit decreasing height* (BFDH) algorithm was first published by Coffman and Shor [21] in 1990, while Lodi et al. [51,53] introduced the *floor-ceiling* (FC) algorithm in 1998. There are four variations of the algorithm, namely the oriented (O) case allowing non-guillotineable (free – F) packing (FC_{OF}), the oriented case where a guillotineable (G) packing is required (FC_{OG}), the case where 90° rotations (R) are allowed and a guillotineable packing is required (FC_{RG}), and the case where rotations and free packing are allowed (FC_{RF}). In these algorithm variations, the items are not only packed from left to right on the floor (the horizontal line defined by the bottom edge of the tallest item on the level), as in all the other algorithms, but also from right to left on the ceiling of the level (the horizontal line defined by the top edge of the tallest item on the level). Bortfeldt [12] improved upon the BFDH algorithm (and named his algorithm the BFDH* algorithm) for the case where rotations are allowed. In order to determine initial solutions for their exact approaches to the strip packing problem, Martello et al. [56] developed an algorithm called *JOIN*. The final known heuristic used for comparison is the *size-alternating stack* (SAS) algorithm by Ntene and Van Vuuren [58,59].

2.2. Variable-sized bin packing

Friesen and Langston [29] introduced a repacking strategy called *first-fit decreasing using largest bins, at end repack to smallest possible bins* (FFDLR) for one-dimensional bin packing. This strategy forms an important component of two-stage algorithms for the variable-sized bin packing problem. In an attempt to minimise the wasted space in bins occupied by items, the strategy is to pack all items into the largest bins first and then to attempt repacking the items in these bins into smaller bins. They also introduced the *first-fit decreasing using largest bins, but shifting as necessary* (FFDLS) strategy. Here, all items in a bin are shifted to the smallest bin that will hold them when items are packed into a bin containing another item larger than $1/3$ of the largest bin in area, such that the total size of the items is greater than or equal to $3/4$ of the size of the smallest bin. This shifting procedure is followed by the repacking strategy of the FFDLR algorithm once all items have been packed.

Kang and Park [47] combined the FFDLR strategy with the *first-fit decreasing* (FFD) and *best-fit decreasing* (BFD) algorithms (by Johnson et al. [45]) to arrive at the *iterative first-fit decreasing* (IFFD) and *iterative best-fit decreasing* (IBFD) algorithms for one-dimensional packing. These algorithms achieve an optimal packing when the sizes of items and bins are exactly divisible (*i.e.* when $w_{j+1}/w_j \in \mathbb{N}$ for all $j = 1, 2, \ldots, n-1$, where w_j is the size of an item or bin).

The two-phase approach to two-dimensional bin packing originates from the *hybrid first-fit* (HFF) algorithm by Chung et al. [17], the *finite best strip* (FBS) algorithm[2] developed by Berkey and Wang [11] and the *hybrid next-fit* (HNF) algorithm by Frenk and Galambos [28]. Lodi et al. [51,53] applied the two-phase approach to their *floor-ceiling* (FC) algorithms to effectively develop the *hybrid floor-ceiling* (HFC) algorithm (although they did not call it such). These hybrid algorithms pack items into a strip using a level algorithm, and then repack the levels into bins using one of the *next-fit decreasing* (by Johnson [44]), FFD or BFD algorithms for one-dimensional bin packing.

3. New and improved strip packing heuristics

In this section we introduce four new and improved algorithms for strip packing in some detail. A brief introduction to each algorithm is followed by a pseudocode listing of the procedure.

[2] As Monaci [57, p. 37] and Lodi et al. [50, p. 246] note, this should be called the *hybrid best-fit* (HBF) algorithm for the sake of uniformity.

Author's personal copy

3.1. Modified size-alternating stack algorithm

While studying the results produced by the SAS algorithm by Ntene and Van Vuuren [58,59], it became clear that some improvements could be made to the algorithm. As for the SAS algorithm, the set of items to be packed is partitioned into two sets in the *modified size-alternating stack* (SASm) algorithm, a set of narrow items $\mathscr{N}$ for which the height is greater than the width ($h_i > w_i$), and a set of wide items $\mathscr{W}$ for which $w_i \geqslant h_i$. Sorting the items in $\mathscr{N}$($\mathscr{W}$) by decreasing height (width) and resolving any equalities by additionally sorting by decreasing width (height) leads to a small improvement on average (these types of sorting are abbreviated as DHDW and DWDH, respectively). Secondly, by searching through the entire set $\mathscr{W}$ for the tallest item and comparing that to the first item in $\mathscr{N}$ (instead of the first item in $\mathscr{W}$) when initialising a level, an additional gain may be made as a new level will not necessarily have to be created if, say, the second item in $\mathscr{W}$ is taller than the first item of both sets. The SAS algorithm allows wide items to be stacked on top of one another and narrow items next to the wide items if there is sufficient space. Define a rectangle of free space above the last wide item bounded vertically by the ceiling of the level and the top of the last wide item in the stack, and horizontally by the width of the same item. It is possible that more space may be utilised than before by filling this space with narrow items in a FFDH manner. The final improvement allows narrow items to be placed next to each other while stacking (the SAS algorithm only allows one narrow item to be stacked onto another). This may be achieved by defining a rectangle of free space above the floor-packed narrow item and filling it with other narrow items in a FFDH manner. Floor-packed items are selected from alternating sets, as in the SAS algorithm. The resulting solution satisfies the guillotine constraint. Algorithm 1 contains a pseudocode listing for the SASm algorithm resulting from the incorporation of these improvements into the original SAS algorithm.

Algorithm 1. Modified size-alternating stack algorithm (SASm)

Input: A list of items to be packed.
Output: A feasible packing of the items.
1: partition the list of items into a list of narrow items ($h_i > w_i$) and a list of wide items ($w_i \geqslant h_i$)
2: sort the narrow items by DHDW and the wide items by DWDH
3: **while** there are unpacked items **do**
4: **if** a level is being initialised **then**
5: compare the tallest of the narrow and wide items
6: initialise the level with the tallest of these
7: **else if** the last floor item was a wide item or no wide items remain **then**
8: pack the first narrow item in the list and define the rectangle of free space
9: fill this rectangle with other narrow items in a FFDH manner
10: **else if** the last floor item was a narrow item or no narrow items remain **then**
11: stack wide items above each other until no space or wide items remain
12: if possible, stack narrow items in the free space next to stacked wide items
13: if possible, stack narrow items in the free space above the last stacked wide item
14: **end if**
15: **end while**

3.2. Best-fit with stacking algorithm

The *best-fit with stacking* (BFS) algorithm is an attempt at further improving Bortfeldt's BFDH* algorithm [12] by implementing the stacking procedure for narrow items in the SASm algorithm for floor-packed items. However, unlike the BFDH* algorithm, the BFS algorithm does not wait until a level is full before packing onto the floor-packed items. Instead, the stacking procedure is attempted when an item is packed onto the floor. Initially, instead of only sorting by decreasing height, the items are sorted by decreasing height and any equalities are resolved by additionally sorting according to decreasing width. When the algorithm packs an item onto the floor, a rectangle is defined vertically by the top of the floor-packed item and the ceiling of the level, and horizontally by the width of the floor-packed item. This rectangle is then filled by unpacked items in a FFDH manner. If the space next to a floor-packed item is smaller than any unpacked item, the width of the rectangle may be increased so that the right-hand boundary of the rectangle is adjacent to the right-hand boundary of the strip (or bin). The resulting solution satisfies the guillotine constraint. Algorithm 2 contains a pseudocode listing for the BFS algorithm.

Algorithm 2. Best fit with stacking algorithm (BFS)

Input: A list of items to be packed.
Output: A feasible packing of the items.
1: sort the items by DHDW
2: **while** there are unpacked items **do**
3: select the first unpacked item in the list
4: find the level with minimum residual horizontal space for that item
5: **if** no level has space for the item **then**
6: initialise a new level with the item
7: **else if** such a level exists **then**
8: pack the item into the level
8: define the rectangle of free space above the item
10: **if** the space next to the item is too small for the narrowest unpacked item **then**
11: expand the region of the rectangle of free space
12: **end if**
13: pack items into the rectangle of free space in a FFDH manner
14: **end if**
15: **end while**

3.3. Ceiling stacking algorithms

Our final two new strip packing heuristics, called the *stack ceiling* (SC) and *stack ceiling with re-sorting* (SCR) algorithms, make use of the idea of packing onto ceilings as in the FC algorithms, and the stacking of items as in the SAS algorithms. However, for these algorithms we do not stack onto items on the floor; instead we stack downwards onto ceiling-packed items. Items are first sorted in order of decreasing height and any equalities are resolved by decreasing width. The entire list of items is sequentially searched to determine whether the items fit onto the floor. Those that fit onto the floor are removed from the list of unpacked items and added to the list of packed items.

When no further items fit onto the floor, as many items as possible are packed onto the ceiling (the SCR algorithm first re-sorts the items according to decreasing width and resolves any equalities by sorting those items by height). This happens in the following recursive manner, starting in the top-right corner, packing downwards, and from right to left. First, the tallest (widest for

Author's personal copy

SCR) item is packed into the top-right corner of the ceiling. Thereafter, the list of unpacked items is searched for a rectangle that may be stacked below the ceiling-packed item. If one is found, it is stacked below the first item and this process continues recursively until no space remains for any further packing. During the process of stepping backwards through the recursion steps, an attempt is made to start a new recursive packing process to the left of the items if enough space remains between the left-hand boundary of an item and the left-hand boundary of the item below it. This process continues until no further items may be packed onto the ceiling. At this point the SCR algorithm re-sorts the items in order of decreasing height, resolving equalities in order of decreasing width. If any items remain, a new level is created and the process is repeated. The resulting solutions from either algorithm may not satisfy the guillotine constraint. Pseudocode listings for these two algorithms may be found in Algorithm 3.

Algorithm 3. Stack ceiling {with re-sorting} algorithm (SC{R})

Input: A list of items to be packed.
Output: A feasible packing of the items.
1: sort the items by DHDW
2: **while** there are unpacked items **do**
3: initialise a level and fill it in a FFDH manner
4: **if** there are unpacked items and sufficient space on the ceiling for these items **then**
5: {*Here SCR would re-sort the items by decreasing width and decreasing height*}
6: **while** there are unpacked items and they fit onto the ceiling **do**
7: recursively stack items onto the ceiling
8: **end while**
9: {*Here SCR would re-sort the items by decreasing height and decreasing with*}
10: **end if**
11: **end while**

Although the SCR algorithm is listed as re-sorting the items at certain stages of the algorithm, such an implementation approach would slow it down considerably. Instead, it is possible to generate a copy of the list of items and sort the copy by decreasing width and decreasing height. By linking the two lists via another data structure, the removal of an item from both lists is possible. This means the algorithm performs only two sorting operations at the cost of little more memory use, instead of two sorting operations per level that is created.

4. A new variable-sized bin packing heuristic

All level strip packing algorithms may be combined with the NFD, FFD or BFD algorithms for one-dimensional bin packing to create hybrid versions of the algorithms for the Single Bin Size Bin Packing Problem. Since the bin width and bin height may not be the same for all bins, as in the Multiple Bin Size Bin Packing Problem, bins are filled in a FFD manner before the next bin in the list is considered and a new strip packing may be required for the new bin width. The level heights resulting from a SAS packing are not guaranteed to be in descending order[3] (in terms of height) from the bottom of a strip upwards. Therefore, the levels are packed into bins in a *first-fit* manner. The above two-phase approach combined with the FFDLR strategy allows for the packing of variable-sized bins with the aim of minimising the total area of the bins containing items. This approach may be labelled a two-dimensional version of the IFFD algorithm introduced by Kang and Park [47] and consists of two stages.

During the first stage, the items are sorted according to decreasing height and decreasing width, and the bins are sorted by decreasing area, with equalities resolved by sorting by increasing perimeter. A strip packing is performed with the strip width taken as the width of the first bin in the list. The levels of the strip are then packed into the bin from the bottom upwards until no further levels fit. If the next bin in the list has the same bin width, the remaining levels are packed into that bin. If there are unpacked levels and the next empty bin has a different width, another strip packing is performed with the remaining items, taking the strip width as the new bin width. This process continues until all items have been packed into bins.

During the second stage, a bin is selected for which the total area of the items in that bin is the smallest over all bins. The list of bins is scanned for the smallest empty bin whose area is greater than or equal to the area of the items in the packed bin. A strip packing is performed for these items, taking the strip width as the width of the empty bin. If the strip height is not greater than the bin height, the items may be packed into the empty bin. The previously packed bin is now empty. However, if the strip height is greater than the bin height, then the suitability of the previous (different) bin in the bin list is investigated. Once an attempt has been made to repack the items, the process is repeated with the bin corresponding to the next largest packed item area. This process continues until all bins have been investigated for repacking. A pseudocode listing of the two-stage algorithm for variable-sized bins (2SVSB) may be found in Algorithm 4.

Algorithm 4. Two-stage algorithm for variable-sized bins (2SVSB)

Input: A list of items to be packed and a list of bins into which these items should be packed.
Output: A feasible packing of the items into the bins with the aim of minimising unutilised space in bins containing items.
1: sort the bins by decreasing area and increasing perimeter
2: **while** there are unpacked items **do**
3: set the strip width equal to the width of the largest empty bin
4: pack all unpacked items into a strip with the same width
5: fill the bin with levels in a FFD manner
6: **end while**
7: **while** there are bins for which a repacking has not been attempted **do**
8: select bin o with smallest item area for which no repacking has been attempted
9: let s be the smallest empty bin with area $\geqslant$ to the area of items in o
10: **while** s is not filled and not equal in size to o **do**
11: perform a packing of items into a strip of width equal to the width of s
12: **if** the strip height is less than the bin height **then**
13: leave the items in bin s and add o to the list of empty bins
14: **else**
15: let the previous (larger) bin in the empty bin list be s
16: **end if**
17: **end while**
18: **end while**

[3] Consider the following situation: A new level is to be initialised and two wide items remain. One item is shorter, but wider than the other. The SAS algorithm would initialise the level with the shorter item, necessitating the creation of a new level of greater height for the taller item.

Author's personal copy

5. Benchmarks

In order to evaluate the effectiveness of the new algorithms for strip packing and variable-sized bin packing described in Sections 3 and 4, respectively, we compare the results obtained by means of these algorithms with results obtained via the known algorithms in Section 2.1 for a large set of benchmark data. The benchmark data used for this purpose are introduced in this section.

A number of repositories of benchmark data for various packing problems are available online. These include the EURO Special Interest Group on Cutting and Packing (ESICUP) repository [26] and the repository for strip packing problems by Van Vuuren and Ortmann [66]. The benchmarks used to evaluate the strip packing algorithms are comprised of the instances by Hopper and Turton [38,40], and Wang and Valenzuela [68].

Although many benchmarks are available for strip or single-sized bin packing problems, there are not many available for the variable-sized bin packing problem. Some industry data are available (such as those supplied by Wang [67, p. 585]), but the benchmark sets by Hopper and Turton [38,42], and Pisinger and Sigurd [60] appear to be the only algorithmically generated benchmark data available for this problem.

We created additional benchmark instances of our own in order to test the effects of the new and improved algorithms on nice[4] and pathological data. Hence, our benchmarks were created in a manner similar to those of Wang and Valenzuela [68] (however, the dimensions are integer values). Our data generation algorithm begins with a square rectangle of dimensions 1000 by 1000. An optimal solution is therefore known in each case. It can create pathological benchmarks, or nice benchmarks. The item dimensions are restricted such that no item height or width is greater than the smallest bin dimension.

Pathological rectangles are split in a manner similar to that by Wang and Valenzuela [68]. However, in an attempt to find valid benchmark instances faster, the cut direction was not always decided randomly. Instead, the dimension that exceeds the minimum bin dimension by the greatest margin was split in two. For pathological data, the difference between the smallest bin dimension and the item's largest dimension was used to determine a cut point that would increase the speed of finding a valid set of items. If a nice data set was required, the point was assigned randomly within limitations discussed in detail by Wang and Valenzuela [68].

Initially, cuts were made into the original rectangle to determine the bin sizes. These bins were subsequently used as initial items, which were further split until the required number of items had been found. A random number of copies of each bin size was created using a uniform distribution in the range [1,6]. Once the required number of items had been cut, the validity of the result was determined. The item and bin sets were only valid if the total area of bins was greater than three times the area of the items and the items could fit into any bin, even if rotated.

We created five copies of each of a combination of 2, 3, 4, 5 and 6 bin sizes, and 25 (not for 6 bin sizes), 50, 100, 200, 300, 400 and 500 items, resulting in 340 benchmark instances for the variable-sized bin packing problem. These benchmarks may be found online [66].

6. Numerical results

The results of the various strip packing algorithms applied to the 571 benchmark instances are presented and analysed in this section. The effectiveness of the algorithms in terms of achieving approximate solutions to the variable-sized bin packing problem are also investigated.

6.1. Strip packing results

The set of 571 benchmark strip packing problem instances listed in Section 5 were used to evaluate the 11 algorithms in Sections 2.1 and 3. The results are shown in Tables 1–3. The strip packing height is measured relative to the optimal packing height in each case. The number of times an algorithm achieved the minimum or maximum strip packing height are shown in rows 8 and 9 of Table 1, respectively. The time data were obtained by running each algorithm for each benchmark seven times, removing the fastest and slowest results and averaging the five remaining iterations. The process of data sorting was included as part of the timing and was performed using the Merge Sort algorithm [46]. The calculations were all performed on a Windows XP PC with a 3.0 GHz Intel Core 2 Duo CPU and 4 GB RAM. The algorithms discussed in Section 2 were all written in Visual Basic .NET by the authors of this paper from their understanding of the algorithms' descriptions in the literature. The other authors' results are from their respective papers.

Comparing the results of the 11 algorithms in Tables 1–3, it is clear that the use of stacking or packing on the ceiling improves the packing density, especially for pathological data. The worst packing height results produced by the SAS algorithms unfortunately result in a greater strip height than obtained by the simpler FFDH and BFDH algorithms. For nice data the older algorithms are slightly better, but for pathological data it is better to use the SAS algorithms. The modified version of the SAS algorithm outperforms the original, both in terms of packing density and completion time for larger data sets (see Table 1). It also achieves the best packing more often (achieved a best unique strip height 9 times) and achieves the largest strip packing height less often. The BFS and FC_{OG} algorithms achieve similar packing densities, with the BFS performing slightly better on pathological data (due to its stacking capability) and FC_{OG} performing marginally better on nice data (due to ceiling items possibly having larger widths than the floor-packed items below). However, the BFS algorithm is superior in terms of time, requiring (on average) approximately half of the time it takes for FC_{OG} to find a similar packing density.

The algorithms that are not required to adhere to the guillotine constraint generally achieve packings that are more dense than those that allow for guillotine cutting, as expected. The SC algorithm is an improvement over the FC_{OF} algorithm as the strip heights are generally lower (especially for pathological data) and the packings are found in less time. The SCR algorithm also generally finds better solutions than the FC_{OF} algorithm (but worse than the SC algorithm) – however, it is the slowest of all the algorithms tested, averaging 6.6 seconds for 5000 items of pathological data. The other two non-guillotine algorithms achieve the best packing more often, but more than three quarters (119 of 157) of the minimum strip height results for the SCR algorithm are unique to it.

A comparison between the new and improved algorithms and recent results in the literature may be found in Tables 2 and 3. The *best-fit* (BF) algorithm by Burke et al. [14] is a plane-packing heuristic shown to yield better results [13] as a decoder for metaheuristics than *bottom-left*-based plane-packing heuristics on the benchmark instances by Hopper and Turton [40]. Note that this algorithm and the HRBB algorithm by Cui et al. [23] allow the items to be rotated by 90°, while the other algorithms listed in Table 2 do not. Burke et al. timed their algorithm on a PC with an 850 MHz CPU and 128 MB RAM [13, p. 664]. The BF algorithms are not forced to pack into levels and are hence expected to pack items more densely than the new algorithms. The other algorithms are metaheuristics and provide better solutions than the heuristics,

[4] Nice data satisfies the area ratio constraint (the largest item is not more than seven times larger than the smallest item) and the aspect ratio constraint $1/4 \leqslant h_i/w_i \leqslant 4$ – the same values used by Wang and Valenzuela [68, p. 387].

Author's personal copy

Table 1
Summary of results for the strip packing problem. The row labelled *Guillotine* indicates whether or not the solutions are guaranteed to be guillotineable and the row labelled *Packing* indicates whether the solution is a level packing (L) or a pseudolevel packing (PL). All algorithms achieved an optimal solution at least once. All times are given in seconds. The notation $a(b)$ in the *Minimum* H and *Maximum* H rows indicates that the minimum/maximum strip height was achieved a times, of which it was achieved by the algorithm uniquely a total of b times.

	FFDH	BFDH	JOIN	FC_{OG}	BFDH*	SAS	SASm	BFS	FC_{OF}	SC	SCR
Source	[20]	[21]	[56]	[51,53]	[12]	[58,59]	New	New	[51,53]	New	New
Packing	L	L	L	PL	PL	PL	PL	PL	PL	PL	PL
Guillotine	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
Ave. H/Opt	132.2%	132.2%	134.2%	118.1%	124.7%	126.2%	123.9%	116.6%	117.3%	112.9%	114.7%
Max. H/Opt	182.7%	182.7%	182.8%	156.1%	176.6%	195.1%	195.1%	169.2%	156.1%	153.2%	153.2%
Ave. Nice	118.3%	118.3%	120.9%	112.7%	114.4%	120.9%	119.7%	114.1%	112.1%	112.0%	112.7%
Ave. Path	146.9%	146.9%	148.2%	124.7%	136.0%	132.5%	128.9%	119.6%	123.8%	114.2%	117.6%
Minimum H	9(0)	9(0)	1(1)	75(0)	26(3)	9(0)	20(8)	47(8)	198(1)	330(128)	157(119)
Maximum H	24(1)	21(0)	240(215)	1(0)	9(0)	122(36)	88(4)	2(0)	1(0)	1(0)	2(0)
Ave. Time t	0.3280	0.3313	0.3700	1.0443	0.7975	0.2837	0.2432	0.3698	0.6968	0.4465	1.0375
Nice 5000 t	2.1370	2.1385	2.1161	4.3850	5.6026	1.2806	0.9921	2.2177	4.3622	2.2831	4.8914
Path 5000 t	2.1393	2.1406	2.1200	5.1266	4.9463	1.3895	1.0916	2.8224	5.0690	2.6315	6.5756

Table 2
A comparison of heuristics with some of the recent results in the literature on the benchmark instances by Hopper and Turton [38,40]. Results are represented as gaps (%) above the optimal solution. The *Alg T* row indicates whether the algorithm is a heuristic (H), metaheuristic (MH) or exact (E) method. The *Packing* row indicates which algorithms result in level (L), pseudolevel (PL) or plane (P) solutions. The *Problem* row indicates the problem type, as defined by Lodi et al. [53]. All times in the rows labelled Ave. t are in seconds.

	FFDH	BFDH	JOIN	FC_{OG}	BFDH*	SAS	SASm	BFS	FC_{OF}	SC	SCR	BF	SPGAL		GRASP	SVC	BS	HRBB
Source	[20]	[21]	[56]	[51]	[12]	[59]	New	New	[51]	New	New	[14]	[12]	[12]	[5]	[9]	[9]	[23]
Alg T	H	H	H	H	H	H	H	H	H	H	H	H	MH	MH	MH	MH	MH	E
Packing	L	L	L	PL	PL	PL	PL	PL	PL	PL	PL	P	P	P	P	P	P	P
Problem	OG	OG	OG	OG	OG	OG	OG	OG	OF	OF	OF	RF	OG	OF	OF	OF	OF	RG
C1	40.0	36.7	40.0	13.3	13.3	36.7	28.3	15.0	13.3	13.3	10.0	11.7	3.2	1.6	0.0	1.7	1.7	1.7
C2	15.6	15.6	20.0	8.9	11.1	20.0	11.1	11.1	6.7	8.9	11.1	13.3	3.3	3.3	0.0	0.0	0.0	0.0
C3	25.6	25.6	28.9	17.8	18.9	23.3	16.7	17.8	17.8	14.4	15.6	10.0	3.9	3.2	1.1	1.1	1.1	1.1
C4	26.7	26.7	32.8	12.8	23.3	20.6	15.6	11.7	11.1	6.1	9.4	5.0	3.8	3.5	1.6	1.7	1.7	2.2
C5	16.3	16.3	25.2	9.6	12.6	12.2	11.1	9.6	7.4	7.0	8.1	4.1	2.4	2.0	1.1	1.1	1.1	1.9
C6	17.2	16.7	18.6	5.6	8.9	13.3	11.1	6.1	5.6	5.3	5.8	3.3	1.9	1.7	1.6	0.8	1.4	1.4
C7	13.5	13.5	14.9	4.9	8.8	10.8	9.0	4.9	4.9	3.9	4.9	2.4	1.7	1.5	1.4	0.8	1.1	1.4
Ave. t	4E−4	5E−4	4E−4	17E−4	10E−4	4E−4	4E−4	6E−4	15E−4	6E−4	11E−4	±0.01	143	159	60	50	50	1.86
T1	46.5	46.5	46.7	33.3	41.9	41.2	41.2	36.8	33.3	30.3	28.5	–	–	–	0.0	0.9	1.0	–
T2	40.8	40.8	44.0	26.1	33.4	34.4	31.2	27.0	24.8	21.8	19.4	–	–	–	3.2	3.5	4.0	–
T3	40.7	40.6	45.4	22.1	32.2	34.6	34.6	22.0	20.0	15.4	15.7	–	–	–	3.7	3.3	4.3	–
T4	27.5	27.5	29.3	12.2	18.6	21.9	23.5	13.9	10.7	8.9	11.9	–	–	–	3.0	2.5	3.2	–
T5	27.2	27.2	29.7	11.6	22.3	17.4	17.4	12.2	9.1	7.8	8.2	–	–	–	2.4	2.1	2.7	–
T6	31.7	31.7	32.5	13.1	21.9	16.6	16.7	9.9	12.4	6.0	8.0	–	–	–	2.1	1.6	2.1	–
T7	27.4	27.4	29.0	10.7	16.6	12.5	10.8	6.3	10.1	5.1	5.6	–	–	–	1.5	1.0	1.4	–
Ave. t	5E−4	5E−4	5E−4	14E−4	12E−4	5E−4	4E−4	7E−4	13E−4	7E−4	12E−4	–	–	–	60	50	50	–
N1	37.5	35.9	38.0	21.1	27.1	27.1	27.1	23.9	20.7	20.2	20.5	–	–	–	0.9	3.3	4.4	–
N2	33.1	33.1	36.2	18.6	21.1	34.7	30.6	19.5	17.2	18.5	17.3	–	–	–	3.3	3.4	4.2	–
N3	36.8	36.8	40.8	19.8	30.4	29.8	28.6	20.3	18.0	16.0	17.2	–	–	–	3.6	3.5	4.2	–
N4	28.6	28.6	30.2	12.8	22.2	22.4	22.5	14.3	10.7	9.6	12.8	–	–	–	3.0	2.5	3.1	–
N5	28.2	28.2	31.3	11.5	24.0	18.9	18.9	12.3	9.1	8.2	8.6	–	–	–	2.6	2.1	2.7	–
N6	28.2	28.1	29.3	12.3	21.1	17.7	17.7	9.8	11.4	5.6	8.5	–	–	–	2.2	1.7	2.2	–
N7	23.9	23.9	25.8	9.0	16.5	14.8	14.4	5.6	8.3	4.5	5.7	–	–	–	1.3	1.0	1.0	–
Ave. t	5E−4	4E−4	5E−4	16E−4	11E−4	4E−4	4E−4	7E−4	13E−4	7E−4	12E−4	–	–	–	60	50	50	–

with the disadvantage of requiring more time. The results for the SPGAL [12] and GRASP [5] algorithms are average values. The results by Cui et al. [23] are not exact, because they introduced relaxations in order to find faster solutions. Bortfeldt [12, p. 829] timed his algorithms on a PC with a 2 GHz CPU, Alvarez-Valdes et al. [5, p. 1075] used a 2 GHz Pentium 4 Mobile CPU, Belov et al. [9, p. 829] used Linux workstations with 2×2.4 GHz CPUs and 4 GB RAM, and Cui et al. [23, p. 1287] used a 2.8 GHz Pentium 4 CPU and 512 MB RAM.

In order to compare the algorithms in terms of both packing density and time, the *strip packing efficiency* I^{SP} is defined as

$$I^{SP}_{\mathtt{A}}(\mathscr{L}) = \frac{\mathtt{OPT}(\mathscr{L})}{\mathtt{A}(\mathscr{L})} \times \left(\frac{\tau_{\mathscr{L}}}{t^{\mathtt{A}}_{\mathscr{L}}}\right)^{\frac{1}{\ell}},$$

where $\ell \in \mathbb{N}$, $\mathtt{OPT}(\mathscr{L})$ is the strip height of an optimal solution to a problem with item list $\mathscr{L}$, $\mathtt{A}(\mathscr{L})$ is the strip height of the solution by algorithm A, $t^{\mathtt{A}}_{\mathscr{L}}$ denotes the time required by algorithm A to find a solution for the items in $\mathscr{L}$ and $\tau_{\mathscr{L}}$ denotes the time required by the fastest algorithm in the comparison group to find a solution to the same problem. The influence of time on I^{SP} decreases as the value of ℓ increases. An algorithm may be labelled as more efficient than a second algorithm for a given value of ℓ if its efficiency is greater. The effect on the algorithmic ranking of changing the parameter ℓ is shown in Table 4.

For low values of ℓ (time is allocated a high importance), the three level-packing and two SAS algorithms perform well due to their fast execution times. With an increase in the value of ℓ, the SC algorithm becomes the highest ranked, while the SAS algorithm and the SASm algorithm remain the closest competition due to their packings typically being more dense than those of the FFDH algorithm and the BFDH algorithm, and their speed in completing a packing. For high values of ℓ, the SC algorithm becomes the highest ranked algorithm, closely followed by the SCR and BFS algorithms. The FFDH and BFDH algorithms trade speed for packing density; hence their rankings drop as ℓ increases. The JOIN algorithm

Author's personal copy

Table 3
Gaps (%) between the average strip packing heights obtained and the optimal packing heights for the strip packing problem on the benchmark instances by Wang and Valenzuela [68].

	FFDH	BFDH	JOIN	FC_{OG}	BFDH*	SAS	SASm	BFS	FC_{OF}	SC	SCR	BF	BF + SA	GRASP
Source	[20]	[21]	[56]	[51]	[12]	[59]	New	New	[51]	New	New	[14]	[14]	[5]
Alg T	H	H	H	H	H	H	H	H	H	H	H	H	MH	MH
Packing	L	L	L	PL	PL	PL	PL	PL	PL	PL	PL	P	P	P
Problem	OG	OG	OG	OG	OG	OG	OG	OG	OF	OF	OF	RF	RF	OF
Nice25	30.8	30.8	34.5	21.1	23.5	31.1	27.7	22.7	20.1	19.8	21.3	8.0	4.0	3.7
Nice50	21.9	21.9	24.8	16.1	18.5	25.2	23.5	17.9	15.6	15.4	16.0	9.7	4.4	4.6
Nice100	17.7	17.7	20.2	11.7	13.7	19.3	18.9	13.4	11.3	11.2	11.8	7.9	5.0	4.0
Nice200	13.1	13.1	15.6	8.9	10.1	15.7	15.6	10.0	8.2	8.2	8.6	6.9	4.7	3.6
Nice500	8.2	8.2	9.7	5.9	6.5	12.5	12.7	6.6	5.4	5.4	5.8	3.4	3.5	2.2
Nice1t	6.1	6.1	6.9	4.5	4.8	11.4	11.4	4.9	4.0	4.0	4.2	3.8	2.9	2.2
Nice2t	4.5	4.5	5.3	3.1	3.3	10.6	10.6	3.5	2.9	2.9	3.1	–	–	–
Nice5t	3.5	3.5	3.8	2.1	2.2	10.1	10.1	2.3	1.9	1.9	2.1	–	–	–
Ave. *t*	0.102	0.102	0.101	0.214	0.269	0.061	0.047	0.106	0.212	0.109	0.232	–	–	60
Path25	48.8	48.8	49.9	28.5	38.4	48.5	46.4	30.4	27.3	23.7	29.5	10.2	3.1	4.2
Path50	49.2	49.2	50.8	25.4	39.5	38.5	35.8	26.1	24.3	16.3	22.8	13.7	3.4	1.8
Path100	49.6	49.6	50.9	23.9	38.1	31.7	29.9	19.4	23.2	12.5	16.6	6.8	3.0	2.6
Path200	47.8	47.8	49.1	23.3	34.5	25.1	18.7	13.0	22.3	9.3	11.4	4.1	3.4	2.0
Path500	42.0	42.0	43.3	22.5	30.1	20.1	12.1	7.6	21.8	8.4	6.8	3.8	3.5	3.1
Path1t	41.9	41.9	42.8	23.6	30.9	18.3	10.8	7.3	23.1	8.6	6.2	3.1	2.9	2.5
Path2t	34.4	34.4	35.1	20.3	25.3	13.6	09.4	5.4	20.0	6.8	4.5	–	–	–
Path5t	30.5	30.5	31.0	20.3	24.3	10.2	07.3	4.5	20.0	6.9	3.6	–	–	–
Ave. *t*	0.102	0.102	0.101	0.251	0.237	0.066	0.052	0.138	0.248	0.127	0.314	–	–	60

Table 4
Rankings of the strip packing algorithms according to strip packing efficiency for various values of the parameter ℓ. Increasing the value of ℓ decreases the effect of time on the efficiency score. The best rank that may be achieved by an algorithm is 1, while 11 is the worst rank.

	FFDH	BFDH	JOIN	FC_{OG}	BFDH*	SAS	SASm	BFS	FC_{OF}	SC	SCR
Time	3	4	6	11	9	2	1	5	8	7	10
$\ell = 1$	4	5	6	11	9	2	1	3	8	7	10
$\ell = 2$	5	6	7	11	9	2	1	3	8	4	10
$\ell = 5$	5	6	8	10	11	4	1	2	7	3	9
$\ell = 10$	7	8	10	9	11	4	3	2	5	1	6
$\ell = 20$	9	10	11	6	8	7	5	2	4	1	3
$\ell = 50$	10	9	11	5	8	7	6	2	4	1	3
H/Opt	10	9	11	5	7	8	6	3	4	1	2

becomes the worst-ranked algorithm as the value of ℓ increases due to its poor solution quality compared to the other algorithms tested here.

6.2. Variable-sized bin packing results

In order to test the effectiveness of the algorithms with respect to the variable-sized bin packing problem, two criteria are required, namely execution time and utilisation. The *utilisation* μ of a packing is simply the total area of the items divided by the area of the bins that contain items, that is

$$\mu = \frac{\text{Total area of items}}{\text{Area of bins containing items}}.$$

If the utilisation achieved by a packing is, say 80%, then 20% of the accumulated bin areas constitutes wasted space. Thus the aim is to maximise the utilisation in order to minimise waste.

A combination of the FFDLR and FFD strategies, and the algorithms described in Section 3 were applied to the two problem instances described by Wang [67], the 15 benchmarks created by Hopper and Turton [38,42], the 500 instances generated by Pisinger and Sigurd [60] and the 340 new benchmarks, *i.e.* for a total of 857 benchmarks. Tables 5–9 contain a summary of the results.

When comparing the times in Table 5 to those in Table 1, there are few surprises. Nice benchmarks are still processed faster than pathological benchmarks. The difference in speed between the FC algorithms and the others becomes more pronounced when applied to the variable-sized bin packing data. The two SAS

Table 5
Average packing times (in seconds) achieved for the variable-sized bin packing problem.

	n	FFDH	BFDH	JOIN	FC_{OG}	BFDH*	SAS	SASm	BFS	FC_{OF}	SC	SCR
Source		[20]	[21]	[56]	[51,53]	[12]	[58,59]	New	New	[51,53]	New	New
Packing		L	L	L	PL	PL	PL	PL	PL	PL	PL	PL
Guillotine		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
Overall	133	0.0131	0.0131	0.0134	0.0311	0.0164	0.0121	0.0118	0.0132	0.0298	0.0133	0.0158
Wang P1	2910	5.226	5.238	5.467	14.212	6.064	4.751	4.751	5.312	13.885	5.235	5.870
Wang P2	905	0.2422	0.2451	0.2418	0.6310	0.2717	0.2459	0.2471	0.2438	0.6103	0.2570	0.2833
Ave. M1	100	0.0014	0.0014	0.0013	0.0035	0.0022	0.0012	0.0012	0.0016	0.0031	0.0016	0.0023
Ave. M2	100	0.0016	0.0017	0.0016	0.0048	0.0028	0.0015	0.0015	0.0018	0.0042	0.0018	0.0028
Ave. M3	150	0.0030	0.0032	0.0031	0.0100	0.0049	0.0025	0.0026	0.0034	0.0087	0.0034	0.0053
Ave. PS	60	0.0064	0.0064	0.0064	0.0074	0.0067	0.0066	0.0064	0.0064	0.0071	0.0065	0.0067
Ave. Nice	231	0.0069	0.0069	0.0068	0.0148	0.0131	0.0058	0.0053	0.0074	0.0138	0.0072	0.0108
Ave. Path	231	0.0080	0.0082	0.0078	0.0325	0.0123	0.0061	0.0059	0.0076	0.0295	0.0082	0.0125

Author's personal copy

Table 6
Average packing utilisations achieved for the variable-sized bin packing problem on the benchmark instances by Pisinger and Sigurd [60].

Class	n	FFDH (%)	BFDH (%)	JOIN (%)	FC_{OG} (%)	BFDH* (%)	SAS (%)	SASm (%)	BFS (%)	FC_{OF} (%)	SC (%)	SCR (%)
I	60	86.3	86.6	83.0	87.3	87.4	79.4	86.3	88.6	87.3	88.4	88.3
II	60	83.7	83.7	80.9	85.2	85.0	80.1	82.2	85.1	85.2	85.4	85.4
III	60	81.1	81.7	76.7	81.9	81.9	69.6	75.7	82.2	81.9	81.7	81.7
IV	60	80.0	80.0	79.1	82.7	82.1	76.0	78.0	82.0	83.0	83.0	81.4
V	60	80.4	81.1	77.6	81.3	81.2	72.6	78.3	81.4	81.3	81.2	81.0
VI	60	79.1	79.3	78.3	80.7	80.5	76.1	77.1	79.5	80.7	80.5	80.5
VII	60	79.9	80.2	79.6	80.8	80.6	74.0	80.4	80.9	80.8	80.8	80.9
VIII	60	80.7	81.1	74.2	81.3	81.2	76.4	79.5	81.6	81.3	81.2	81.2
IX	60	72.8	72.6	72.1	72.6	72.8	71.6	72.9	73.0	72.6	73.2	73.2
X	60	83.3	83.8	79.3	85.4	84.6	73.2	79.4	85.5	85.4	85.3	85.1
20	20	73.5	73.7	71.7	75.4	75.4	70.0	72.3	75.2	75.4	75.6	75.2
40	40	79.1	79.3	76.9	80.3	79.8	74.6	78.0	80.1	80.3	80.3	80.0
60	60	81.7	82.0	79.2	83.0	82.7	74.9	80.0	83.1	83.1	83.3	83.2
80	80	84.0	84.1	80.9	84.6	84.4	76.7	81.6	85.0	84.6	84.8	84.7
100	100	85.3	85.8	81.9	86.3	86.3	78.3	83.1	86.5	86.3	86.4	86.3

Table 7
Average packing utilisations achieved for the variable-sized bin packing problem on the new benchmark instances.

Class	n	FFDH (%)	BFDH (%)	JOIN (%)	FC_{OG} (%)	BFDH* (%)	SAS (%)	SASm (%)	BFS (%)	FC_{OF} (%)	SC (%)	SCR (%)
Nice25i	25	73.9	73.6	70.6	73.9	73.6	68.3	71.8	73.6	73.9	74.2	74.2
Nice50i	50	76.7	76.7	73.3	77.9	77.7	70.8	73.1	77.8	77.9	78.4	78.2
Nice100i	100	79.4	79.4	77.5	79.9	79.4	75.7	76.3	79.4	79.9	80.1	80.0
Nice200i	200	82.0	82.0	81.5	84.5	84.5	78.5	79.4	84.5	84.6	85.0	84.8
Nice300i	300	85.8	85.8	83.3	86.0	86.5	80.2	81.7	86.8	86.8	87.2	86.8
Nice400i	400	85.1	85.1	82.7	86.6	85.7	80.2	81.0	85.7	86.6	86.7	86.7
Nice500i	500	87.2	87.2	84.8	87.7	87.7	81.8	83.8	87.7	87.7	87.9	87.7
Path25i	25	76.3	76.3	73.9	77.9	77.6	72.2	74.4	78.3	77.9	78.0	78.0
Path50i	50	76.4	78.6	74.2	81.6	79.4	72.4	75.6	81.9	81.6	79.9	79.9
Path100i	100	79.7	79.7	77.8	83.2	81.3	72.4	75.8	83.5	83.3	83.4	83.4
Path200i	200	84.1	84.0	81.8	88.0	85.9	77.7	79.0	87.5	88.0	88.1	87.9
Path300i	300	82.9	82.9	82.7	87.0	86.0	81.4	81.7	87.3	87.8	87.6	87.7
Path400i	400	82.7	82.7	82.3	89.6	87.0	79.9	80.1	88.5	89.6	89.9	89.6
Path500i	500	82.9	82.9	81.2	88.7	86.4	81.3	82.4	86.7	88.7	88.7	88.6
Nice2bin	225	75.0	75.0	73.4	75.8	75.8	71.3	72.0	75.8	75.8	75.8	75.8
Nice3bin	225	81.5	81.5	79.1	82.8	82.8	76.3	77.7	82.9	83.4	83.5	83.5
Nice4bin	225	83.2	83.0	80.9	83.8	83.4	78.4	81.3	83.6	83.8	84.3	83.9
Nice5bin	225	83.8	83.8	80.7	85.0	84.4	77.7	79.1	84.6	85.0	85.3	85.2
Nice6bin	258	85.3	85.3	83.2	86.2	86.3	80.5	82.1	86.0	86.3	86.8	86.4
Path2bin	225	69.7	70.2	68.5	74.7	71.6	70.8	70.6	74.0	74.8	74.4	74.4
Path3bin	225	80.1	80.1	78.9	86.0	83.9	75.2	76.3	85.2	86.3	86.4	86.3
Path4bin	225	82.9	83.3	81.3	86.6	85.5	74.7	78.3	86.3	86.6	86.2	86.5
Path5bin	225	84.3	84.8	83.2	89.5	88.0	79.7	81.9	89.8	89.8	89.3	89.0
Path6bin	258	88.3	88.4	85.4	90.5	89.5	85.2	86.9	90.5	90.5	90.8	90.6

algorithms remain the fastest, with the BFS algorithm slightly slower than the FFDH and BFDH algorithms.

The comparison of nice and pathological data in Table 7 shows a clear advantage for the pseudolevel algorithms with respect to pathological data over the nice data, with the SAS algorithms being the exceptions. There is also an advantage to having more bin sizes available for packing. There is a clear increase in bin utilisation as the number of bin sizes increases.

Table 8 shows that the non-guillotine algorithms perform better than the algorithms that adhere to the guillotine constraint, as expected. Next best are the FC_{OG}, BFDH* and BFS algorithms, followed by the FFDH and BFDH algorithms, with the SAS algorithms performing worst, on average. This stands in contrast to what was observed in the strip packing results, where the SAS algorithms utilised space better than the level-packing algorithms, on average.

In order to compare the algorithms in terms of both utilisation and time, the *variable-sized bin packing efficiency* Γ^{VS} is defined as

$$\Gamma^{VS}_{A}(\mathscr{B},\mathscr{L}) = \mu^{A}(\mathscr{B},\mathscr{L}) \times \left(\frac{\tau_{\mathscr{L}}}{t^{A}_{\mathscr{L}}}\right)^{\frac{1}{\ell}}.$$

The value of ℓ may be increased in order to reduce the effect of execution time on the efficiency. When this is applied to overall utilisation and average time (as shown in Table 9), the SASm, SAS and BFS algorithms are ranked best due to their fast execution times (and dense packings in the case of the BFS algorithm), followed closely by the SC algorithm due to dense packing and the FFDH and BFDH algorithms due to their speed. The three next best procedures are the SC algorithm (due to its dense packings), the SASm algorithm and the SAS algorithm (due to their fast execution times). The FC and SCR algorithms perform badly according to this measure of efficiency due to the time required to perform packing operations. However, as the time becomes less important (ℓ is increased), the FC and SCR algorithms achieve better rankings (due to their higher packing densities), while the rankings of the SAS algorithms worsen.

7. Conclusions

We proposed four new and improved level-packing algorithms for the strip packing problem in this paper and showed how these algorithms may be employed to solve the variable-sized bin packing problem. The SASm algorithm is a clear improvement over the original in terms of solution time and quality, while the BFS algorithm matches the FC_{OG} algorithm in terms of solution quality, but surpasses it in terms of solution time. The SC and SCR

Author's personal copy

Table 8
A summary of the average packing utilisations achieved for the variable-sized bin packing problem.

	n	FFDH	BFDH	JOIN	FC_{OG}	BFDH*	SAS	SASm	BFS	FC_{OF}	SC	SCR
Source		[20]	[21]	[56]	[51,53]	[12]	[58,59]	New	New	[51,53]	New	New
Packing		L	L	L	PL	PL	PL	PL	PL	PL	PL	PL
Guillotine		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
Overall	133	81.1%	81.4%	78.4%	82.9%	82.4%	75.9%	79.0%	82.9%	83.0%	83.1%	83.0%
Wang P1	2910	84.4%	84.4%	73.4%	84.4%	84.4%	76.8%	77.7%	84.4%	84.4%	84.4%	84.4%
Wang P2	905	87.4%	87.4%	80.1%	90.0%	90.0%	89.6%	89.6%	90.0%	90.0%	90.0%	90.0%
M1	100	93.5%	93.5%	86.8%	94.9%	94.9%	83.5%	91.6%	95.5%	94.9%	95.5%	95.5%
M2	100	87.5%	88.8%	82.8%	89.6%	89.6%	81.7%	88.0%	90.0%	89.6%	90.8%	90.8%
M3	150	92.0%	92.6%	85.5%	93.6%	93.6%	86.8%	91.8%	94.9%	93.6%	93.9%	93.9%
Ave. PS	60	80.7%	81.0%	78.0%	81.9%	81.7%	74.9%	79.0%	82.0%	81.9%	82.1%	81.9%
Ave. Nice	231	81.8%	81.7%	78.6%	82.7%	82.5%	76.8%	78.4%	82.6%	82.8%	83.1%	82.9%
Ave. Path	231	80.9%	81.2%	78.7%	85.3%	83.5%	76.9%	78.5%	85.0%	85.5%	85.3%	85.2%

Table 9
Rankings of the variable-sized bin packing algorithms according to variable-sized bin packing efficiency for various values of the parameter ℓ. Increasing the value of ℓ decreases the effect of time on the efficiency score. The best rank that may be achieved by an algorithm is 1, while 11 is the worst rank.

	FFDH	BFDH	JOIN	FC_{OG}	BFDH*	SAS	SASm	BFS	FC_{OF}	SC	SCR
Time	3	4	7	11	9	2	1	5	10	6	8
$\ell = 1$	5	6	7	11	9	2	1	3	10	4	8
$\ell = 2$	5	4	7	11	9	6	1	3	10	2	8
$\ell = 5$	4	3	8	11	7	9	5	2	10	1	6
$\ell = 10$	5	4	8	11	6	10	7	2	9	1	3
$\ell = 20$	6	5	10	8	4	11	9	2	7	1	3
$\ell = 50$	8	7	10	6	4	11	9	2	5	1	3
μ	8	7	10	4	6	11	9	5	2	1	3

algorithms typically generate more dense solutions than the FC_{OF} algorithm for pathological data, in less time. The use of two linked item lists (each sorted in a different manner) in the SCR algorithm implementation significantly improves its solution time, allowing it to compete with the FC_{OF} algorithm in terms of solution time and packing density. When the items are re-sorted for each level, the algorithm is found to be 20 times slower, on average, than the SC algorithm for 5000 nice items.

There is certainly an advantage to stacking items upwards or downwards within levels, or placing them on the ceiling. For item sets where the sizes vary by a great margin (pathological data), the stacking algorithms (excluding the SAS algorithms) have a clear advantage over those that do not perform a stacking function. The advantage is not as marked for nice data, as expected.

The new strip packing algorithms are not only useful for producing approximate solutions to the variable-sized bin packing problem, but may also be used as decoders for metaheuristics (which outperform these heuristics in terms of solution quality). The non-guillotine pseudolevel algorithms are at a disadvantage compared to plane algorithms due to the restriction of packing into levels, but this does make it possible to use them in a hybrid manner to pack single-size bins. The BFS or FC_{OG} algorithms may be particularly useful if a guillotine restricted packing is required, as various versions of the bottom-left algorithm (which does not guarantee a guillotineable solution) have often been used (see, for example, Jakobs [43] and Hopper and Turton [38,40,41]).

Acknowledgements

The authors wish to thank the anonymous reviewers for their valuable contributions towards improving the presentation of the paper. Work towards this paper was financially supported by the Third World Organisation for Women in Science (TWOWS) in the form of a bursary for the second author, and by the South African National Research Foundation (NRF) in the form of a bursary (GUN 2072815) for the first author and a research grant (GUN 2072999) awarded to the last author.

References

[1] R. Alvarez-Valdes, A. Parajon, J.M. Tamarit, A computational study of LP-based heuristic algorithms for two-dimensional guillotine cutting stock problems, OR Spectrum 24 (2) (2002) 179–192.
[2] R. Alvarez-Valdes, A. Parajon, J.M. Tamarit, A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems, Computers and Operations Research 29 (7) (2002) 925–947.
[3] R. Alvarez-Valdes, F. Parreño, J.M. Tamarit, A grasp algorithm for constrained two-dimensional non-guillotine cutting problems, Journal of the Operational Research Society 56 (4) (2005) 414–425.
[4] R. Alvarez-Valdes, F. Parreño, J.M. Tamarit, A tabu search algorithm for a two-dimensional non-guillotine cutting problem, European Journal of Operations Research 183 (3) (2007) 1167–1182.
[5] R. Alvarez-Valdes, F. Parreño, J.M. Tamarit, Reactive grasp for the strip-packing problem, Computers and Operations Research 35 (4) (2008) 1065–1083.
[6] B.S. Baker, D.J. Brown, H.P. Katseff, A 5/4 algorithm for two-dimensional packing, Journal of Algorithms 2 (4) (1981) 348–368.
[7] B.S. Baker, E.G. Coffman, R.L. Rivest, Orthogonal packings in two dimensions, SIAM Journal on Computing 9 (4) (1980) 846–855.
[8] B. Beisiegel, J. Kallrath, Y. Kochetov, A. Rudnev, Simulated annealing based algorithm for the 2D bin packing problem with impurities, in: H.D. Haasis, H. Kopfer, J. Schönberger (Eds.), Operations Research Proceedings 2005, Springer, Heidelberg, 2006, pp. 309–314.
[9] G. Belov, G. Scheithauer, E.A. Mukhacheva, One-dimensional heuristics adapted for two-dimensional rectangular strip packing, Journal of the Operational Research Society 59 (6) (2008) 823–832.
[10] S. Benati, An algorithm for a cutting stock problem on a strip, Journal of the Operational Research Society 48 (3) (1997) 288–294.
[11] J.O. Berkey, P.Y. Wang, Two-dimensional finite bin-packing algorithms, Journal of the Operational Research Society 38 (5) (1987) 423–429.
[12] A. Bortfeldt, A genetic algorithm for the two-dimensional strip packing with rectangular pieces, European Journal of Operational Research 172 (3) (2006) 814–837.
[13] E.K. Burke, G. Kendall, G. Whitwell, A new placement heuristic for the orthogonal stock-cutting problem, Operations Research 52 (4) (2004) 655–671.
[14] E.K. Burke, G. Kendall, G. Whitwell, A new placement heuristic for the orthogonal stock-cutting problem, Technical Report, School of Computer Science and Information Technology, University of Nottingham, Nottingham, 2006.
[15] B. Chazelle, The bottom-left bin packing heuristic: An efficient implementation, IEEE Transactions on Computers 32 (8) (1983) 697–707.
[16] N. Christofides, C. Whitlock, An algorithm for two-dimensional cutting problems, Operations Research 25 (1) (1977) 31–44.
[17] F.R.K. Chung, M.R. Garey, D.S. Johnson, On packing two-dimensional bins, SIAM Journal on Algebraic and Discrete Methods 3 (1) (1982) 66–76.
[18] F. Clautiaux, J. Carlier, A. Moukrim, A new exact method for the two-dimensional bin-packing problem with fixed orientation, Operations Research Letters 35 (3) (2007) 357–364.

Author's personal copy

[19] E.G. Coffman, M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing: A survey, in: D.S. Hochbaum (Ed.), Approximation Algorithms for NP-hard Problems, PWS Publishing Company, Boston, MA, 1996, pp. 46–93.
[20] E.G. Coffman, M.R. Garey, D.S. Johnson, R.E. Tarjan, Performance bounds for level-oriented two-dimensional packing algorithms, SIAM Journal on Computing 9 (4) (1980) 808–826.
[21] E.G. Coffman, P.W. Shor, Average-case analysis of cutting and packing in two dimensions, European Journal of Operational Research 44 (2) (1990) 134–144.
[22] Y. Cui, Heuristic and exact algorithms for generating homogenous constrained three-staged cutting patterns, Computers and Operations Research 35 (1) (2008) 212–225.
[23] Y. Cui, Y. Yang, X. Cheng, P. Song, A recursive branch-and-bound algorithm for the rectangular guillotine strip packing problem, Computers and Operations Research 35 (4) (2008) 1281–1291.
[24] H. Dyckhoff, A typology of cutting and packing problems, European Journal of Operational Research 44 (2) (1990) 145–159.
[25] K. Eisemann, The trim problem, Management Science 3 (3) (1957) 279–284.
[26] ESICUP, Listing Gallery: Data Sets 2D – Rectangular, Cited: 2007-10-29, 2007. <http://paginas.fe.up.pt/~esicup/tiki-list_file_gallery.php?galleryId=3>.
[27] L. Faina, An application of simulated annealing to the cutting stock problem, European Journal of Operational Research 114 (3) (1999) 542–556.
[28] J.B.G. Frenk, G. Galambos, Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem, Computing 39 (3) (1987) 201–217.
[29] D.K. Friesen, M.A. Langston, Variable-sized bin packing, SIAM Journal on Computing 15 (1) (1986) 222–230.
[30] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting-stock problem, Operations Research 11 (6) (1961) 849–859.
[31] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting-stock problem – Part II, Operations Research 9 (6) (1963) 863–888.
[32] P.C. Gilmore, R.E. Gomory, Multistage cutting problems of two and more dimensions, Operations Research 13 (1) (1965) 94–120.
[33] M. Girkar, B. MacLeod, R. Moll, Performance bound for bottom-left guillotine packing of rectangles, Journal of the Operational Research Society 43 (2) (1992) 169–175.
[34] I. Golan, Performance bounds for orthogonal oriented two-dimensional packing algorithms, SIAM Journal on Computing 10 (3) (1981) 571–582.
[35] J.F. Gonçalves, M.G.C. Resende, A hybrid heuristic for the constrained two-dimensional non-guillotine orthogonal cutting problem, Technical Report, AT&T Labs Research, Florham Park, NJ, 2006.
[36] M. Gradišar, G. Resinovič, M. Kljajić, Evaluation of algorithms for one-dimensional cutting, Computers and Operations Research 29 (9) (2002) 1207–1220.
[37] M. Hifi, V. Zissimopoulos, Constrained two-dimensional cutting: An improvement of Christofides and Whitlock's exact algorithm, Journal of the Operational Research Society 48 (3) (1997) 324–331.
[38] E. Hopper, Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods, Ph.D. Thesis, University of Wales, Cardiff, 2000.
[39] E. Hopper, B.C.H. Turton, A genetic algorithm for a 2D industrial packing problem, Computers in Engineering 37 (1) (1999) 375–378.
[40] 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.
[41] 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.
[42] E. Hopper, B.C.H. Turton, Problem generators for rectangular packing problems, Studia Informatica Universalis 2 (1) (2002) 123–136.
[43] S. Jakobs, On genetic algorithms for the packing of polygons, European Journal of Operational Research 88 (1) (1996) 165–181.
[44] D.S. Johnson, Near-optimal bin packing algorithms, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1973.
[45] D.S. Johnson, A. Demars, J.D. Ullman, M.R. Garey, R.L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms, SIAM Journal on Computing 3 (4) (1974) 295–325.
[46] R. Johnsonbaugh, M. Schaefer, Algorithms, Pearson Prentice-Hall, Upper Saddle River, NJ, 2004.
[47] J. Kang, S. Park, Algorithms for the variable sized bin packing problem, European Journal of Operational Research 147 (2) (2003) 365–372.
[48] L.V. Kantorovich, Mathematical methods of organizing and planning production, Management Science 6 (4) (1960) 366–422 (reprint and translation of 1939 original manuscript).
[49] D. Liu, H. Teng, An improved BL-algorithm for genetic algorithms of the orthogonal packing of rectangles, European Journal of Operational Research 112 (2) (1999) 413–419.
[50] A. Lodi, S. Martello, M. Monaci, Two-dimensional packing problems: A survey, European Journal of Operational Research 141 (2) (2002) 241–252.
[51] 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.), Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Boston, MA, 1998, pp. 125–139.
[52] A. Lodi, S. Martello, D. Vigo, Approximation algorithms for the oriented two-dimensional bin packing problem, European Journal of Operational Research 112 (1) (2002) 158–166.
[53] A. Lodi, S. Martello, D. Vigo, Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, INFORMS Journal on Computing 11 (4) (1999) 345–357.
[54] A. Lodi, S. Martello, D. Vigo, Recent advances on two-dimensional bin packing problems, Discrete Applied Mathematics 123 (1) (2002) 379–396.
[55] A. Lodi, S. Martello, D. Vigo, Models and bounds for two-dimensional level packing problems, Journal of Combinatorial Optimization 8 (3) (2004) 363–379.
[56] S. Martello, M. Monaci, D. Vigo, An exact approach to the strip-packing problem, INFORMS Journal on Computing 15 (3) (2003) 310–319.
[57] M. Monaci, Algorithms for packing and scheduling problems, Ph.D. Thesis, Università di Bologna, Bologna, 2001.
[58] N. Ntene, An algorithmic approach to the 2D oriented strip packing problem, Ph.D. Thesis, University of Stellenbosch, Stellenbosch, 2007.
[59] N. Ntene, J.H. Van Vuuren, A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem, Discrete Optimization 6 (2) (2009) 174–188.
[60] D. Pisinger, M. Sigurd, The two-dimensional bin packing problem with variable bin sizes and costs, Discrete Optimization 2 (2) (2005) 154–167.
[61] D. Pisinger, M. Sigurd, Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem, INFORMS Journal on Computing 19 (1) (2007) 35–36.
[62] V. Pureza, R. Morabito, Some experiments with a simple tabu search algorithm for the manufacturer's pallet loading problem, Computers and Operations Research 33 (3) (2006) 804–819.
[63] D.D.K.D.B. Sleator, A 2.5 times optimal algorithm for packing in two dimensions, Information Processing Letters 10 (1) (1980) 37–40.
[64] P.E. Sweeney, E.R. Paternoster, Cutting and packing problems: A categorized, application-oriented research bibliography, Journal of the Operational Research Society 43 (7) (1992) 691–706.
[65] C.L. Valenzuela, P.Y. Wang, Heuristics for large strip packing problems with guillotine patterns: An empirical study, in: Metaheuristic International Conference: Porto, Cited: 2008-04-22, 2001. <http://users.cs.cf.ac.uk/C.L.Mumford/papers/binpaper.pdf>.
[66] J.H. Van Vuuren, F.G. Ortmann, Benchmarks, under "Benchmarks", Cited: 2008-09-10, 2008. <http://www.vuuren.co.za/main.php>.
[67] P.Y. Wang, Two algorithms for constrained two-dimensional cutting stock problems, Operations Research 31 (3) (1983) 573–586.
[68] P.Y. Wang, C.L. Valenzuela, Data set generation for rectangular placement problems, European Journal of Operational Research 134 (2) (2001) 378–391.
[69] G. Wäscher, H. Haußner, H. Schumann, An improved typology of cutting and packing problems, European Journal of Operational Research 183 (3) (2007) 1109–1130.